98 P 2825

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 198 15 865 A 1**

(51) Int. Cl.[6]: G 06 F 9/45

(21) Aktenzeichen: 198 15 865.3
(22) Anmeldetag: 8. 4. 98
(43) Offenlegungstag: 10. 12. 98

B6

DE 198 15 865 A 1

(30) Unionspriorität:
827619 09. 04. 97 US

(71) Anmelder:
Ricoh Co., Ltd., Tokio/Tokyo, JP

(74) Vertreter:
Schwabe, Sandmair, Marx, 81677 München

(72) Erfinder:
Greenbaum, Jack E., Menlo Park, Calif., US; Baxter, Michael A., Menlo Park, Calif., US

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Kompiliersystem und Verfahren zum rekonfigurierbaren Rechnen

(57) Die Erfindung betrifft ein Kompiliersystem und ein Verfahren zur Erzeugung einer Folge von Programmbefehlen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit, die eine interne Hardwareorganisation aufweist, die wahlweise unter einer Anzahl von Hardwarearchitekturen verändert werden kann, wobei jede Hardwarearchitektur Befehle von einem entsprechenden Befehlsatz ausführt. Quelldateien werden zur Ausführung mit Hilfe von mehreren Befehlsatzarchitekturen (instruction set architectures) kompiliert, wie dies durch Rekonfigurations-Betriebsanweisungen spezifiziert wird. Die Objektdateien fassen wahlweise Bitströme, die Hardwarearchitekturen spezifizieren, die Befehlsatzarchitekturen entsprechen, mit ausführbarem Code zur Ausführung auf den Architekturen zusammen.

DE 198 15 865 A 1

Beschreibung

Die vorliegende Erfindung betrifft generell Software für rekonfigurierbare Computer und insbesondere ein Kompiliersystem und ein Verfahren zur Erzeugung ausführbarer Dateien zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit, die eine veränderbare interne Hardwareorganisation aufweist.

Im Stand der Technik sind Versuche unternommen worden, rekonfigurierbare Geräte bzw. Einheiten zu schaffen. Ein erster bekannter Lösungsansatz besteht in herunterladbaren (down-loadable) Mikrocode-Geräten, bei denen das Verhalten bzw. der Betriebsablauf von festen, nicht rekonfigurierbaren Hardwarebetriebsmitteln zur Ausführung von Programmen wahlweise verändert werden kann, in dem eine bestimmte Version des Mikrocodes (Programmiersprache für ein Steuerwerk) verwendet wird, der in einen programmierbaren Steuerspeicher geladen wird. Ein Beispiel hierfür findet sich in J.L. Hennessy und D.A. Paterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1990. In einigen solcher Systeme kann der Mikrocode vom Benutzer nach der Herstellung geschrieben oder verändert werden. Siehe beispielsweise W.T. Wilner "Design of the Burroughs B1700" in AFIPS Fall Joint Computer Conference, AFIPS Press, 1972; W.G. Matheson, "User Microprogrammability in the HP-21MX Minicomputer", in Proceedings of the Seventh Annual Microprogramming Workshop, IEEE Computer Society Press, 1974. Weil die zugrundeliegende Computerhardware in solchen bekannten Systemen nicht selbst rekonfigurierbar ist, zeigen solche Systeme keine optimierte Rechenleistung, wenn man an einen großen Bereich von Problemtypen denkt. Insbesondere sind solche Systeme generell nicht in der Lage, den Datenpfad zu ändern, sind durch die Größe bzw. Speichergröße der Ausführungseinheiten begrenzt und sind nur in der Lage, Wechsel-Befehlssätze (alternate instruction sets) für dieselbe Hardware zu schaffen. Solche Systeme stellen keinen Einzelkompiler zur Verfügung, der in der Lage ist, zwei verschiedene Architekturen zu kompilieren bzw. zu übersetzen.

Ein zweiter bekannter Lösungsansatz beinhaltet ein System, bei dem die Hardware, die einen Rechenvorgang ausführt, mit Hilfe einer programmierbaren Logik realisiert wird. Es gibt Beispiele hierfür, die feldprogrammierbare Logikschaltungen bzw. feldprogrammierbare Gatetrarrays (FPGAs) von der Stange verwenden (PAM, SPLASH, VCC) und anwenderprogrammierbare Logik (TERAMAC). Siehe beispielsweise: P. Bertin et al., Programmable Active Memories: A Performance Assessment,, Tech. Rep. 24, Digital Paris Research Laboratory, März 1993; D.A. Buell et al., Splash 2: FPGAs in a Custom Computing Machine, IEEE Computer Society Press, 1996; S. Casselman, "Virtual Computing and The Virtual Computer", in IEEE Symposium on FPGAs for Custom Computing Macines, IEEE Computer Society Press, 1994; R. Amerson et al., "Teramac-Configurable Custom Computing", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1995. Im allgemeinen erfordern es diese Technologien, daß eine Anwendung bzw. Applikation hinsichtlich der Hardwarebeschreibung spezifiziert wird, was entweder in Form einer schematischen Beschreibungssprache oder unter Verwendung einer Hardware-Beschreibungssprache, wie beispielsweise VHDL, erfolgt, anstatt daß man Software für einen Computer schreibt, der durch FPGAs festgelegt wird. Beispielsweise wird ein PAM programmiert, indem man ein C++-Programm schreibt, das eine Netzliste erzeugt, die die Konfiguration und Architektur der Gatter bzw. Schaltelemente (Gate) beschreibt. Ein Applikationsentwickler spezifiziert eine Datenstruktur, die eine Hardwarebeschreibung zur Umsetzung der Applikation beschreibt, anstatt daß er eine Spezifizierung eines Applikationsalgorithmusses kompiliert. SPLASH wird auf eine der folgenden drei Arten programmiert: 1) Ein schematisches Erfassungspaket (schematic capture package) zum Aufbauen einer Hardwarespezifizierung, basierend auf einem Schaltschema bzw. schematischen Diagramm; 2) einer Hardware-Beschreibungssprache (wie beispielsweise VHDL), die mit einem Synthese-Paket gekoppelt ist, das VHDL in einfache Gate-Anweisungen (primitives) übersetzt; oder 3) einer DBC, d. h. einer C-Untersprache, die in Gate-Beschreibungen kompiliert wird. TERAMAC wird mit Hilfe eines schematischen Erfassungspakets oder einer Hardware-Beschreibungssprache programmiert. Keines dieser Programmierverfahren beschreibt Algorithmenschritte; statt dessen sorgen sie für einen Mechanismus zur Spezifizierung von Hardware-Architekturen.

Ein dritter bekannter Lösungsansatz beinhaltet rekonfigurierbare Computer, die Softwareprogramme ausführen. Der RISC-4005-Prozessor und der Hokie-Prozessor realisieren Standard-Mikroprozessoren innerhalb von FPGAs. Der RISC 4005 ist im wesentlichen die Demonstration einer eingebetteten (embedded) zentralen Prozessoreinheit (CPU) innerhalb eines kleinen Abschnittes eines FPGAs, dessen weitere Betriebsmittel (resources) einigen Coprozessor-Funktionen zugeordnet sind. Hokie wird als Lernübung für Informatikstudenten oder Elektrotechnikstudenten verwendet. Eine Befehlsatzarchitektur (instruction set architecture; ISA) wird vor der Kompilierung und Ausführung des Programmes ausgewählt und diese Befehlsatzarchitektur wird dann fortwährend verwendet. Außerdem wird der Bitstrom für den Prozessor separat von der Software abgespeichert, die dieser ausführt. Ad hoc-Verfahren werden verwendet, um sicherzustellen, daß ein korrekter Bitstrom geladen wird. Siehe beispielsweise P. Athanas und R. Hudson, "Using Rapid Prototyping to Teach the Design of Complete Computing Solutions", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996. Diese Systeme sorgen nicht für eine Laufzeit-Rekonfigurierung (während der Ausführung des Programm).

Bei einem weiteren bekannten, rekonfigurierbaren Computer handelt es sich um den dynamischen Befehlsatz-Computer (Dynamic Instruction Set Computer; DISC), der eine rekonfigurierbare Prozessoreinheit verwendet. Siehe beispielsweise M.J. Wirthlin und B.L. Hutchings, "A Dynamic Instruction Set Computer", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1995; D.A. Clark und B.L. Hutchings, "The DISC Programming Environment", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996. Die Ausführung und Konfiguration der FPGA der DISC-Prozessoreinheit wird mit Hilfe eines Mikrocontrollers gesteuert, der ebenfalls in Form eines FPGAs realisiert ist. Der Mikrocontroller wird in einem Dialekt der C-Programmiersprache programmiert. Der Kompiler bzw. das Übersetzungsprogramm für diesen C-Dialekt erkennt, daß gewisse Programmanweisungen durch entsprechende Hardware-Konfigurationen der Verarbeitungseinheit ausgeführt werden sollen und sendet einen Mikrocontroller-Code aus, der veranlaßt, daß der richtige Konfigurations-Bitstrom während der Ausführung des Programms in die Verarbeitungseinheit geladen wird. Der Fachmann auf diesem Gebiet wird erkennen, daß der Mikrocontroller seinerseits einen festen Befehlssatz aufweist und daß der Kompiler diesen festen Befehls-

satz kompiliert bzw. übersetzt. Es bestehen mehrere Nachteile hinsichtlich dieser von einem DISC verwendeten Architektur. Weil der Mikrocontroller fest bzw. statisch ist, kann dieser nicht optimiert werden, um verschiedene Arten von Verarbeitungseinheiten zu steuern. Die Konfigurations-Bitströme werden in externer Hardware außerhalb des Speicherplatzes des Mikrocontrollers gespeichert, weshalb das System nicht selbst-enthaltend ist. Außerdem offenbaren die vorstehend im Wege der Bezugnahme in dieser Offenbarung mit beinhalteten Dokumente nicht, wie ein DISC zum Parallelrechnen, zur globalen Signalisierung bzw. systemweiten Datenübermittlung und zum Takten oder zur Handhabung von Interrupts bzw. Unterbrechungsanweisungen verwendet werden könnte. Schließlich werden neue Befehle nur als Einzelgrößen spezifiziert. Der Kompiler sendet nur Befehle für einen Befehlssatz aus, läßt es jedoch zu, daß einzelne Befehle vom Programmierer hinzugefügt werden. Jede Konfigurierung der Verarbeitungseinheit ist ein einzelner Befehl in Form von Hardware, der von dem Programmierer zur Verfügung gestellt wird, wodurch die mögliche Flexibilität eingeschränkt wird.

Ein vierter bekannter Lösungsansatz besteht darin, Systeme zu mischen, wobei verschiedene Teile des Algorithmus bzw. Rechenvorgangs auf verschiedene Komponenten bzw. Elemente des Systems abgebildet werden. Ein bekanntes System bildet einen Algorithmus, der in einem erweiterten C-Dialekt ausgedrückt ist, auf eine gemischte FPGA/DSP-Architektur ab. Der Benutzer markiert ausdrücklich Abschnitte des Eingabeprogramms zur Zuordnung zum DSP, während der Rest des Codes in Gates zur FPGA-Realisierung hinein kompiliert wird. Solche Systeme erfordern spezialisierte Werkzeuge bzw. Tools, weil sie eine nicht übliche Syntax für ISA-Änderungen verwenden. Außerdem ist der Betrieb solcher Systeme mühsam, was an der Verwendung von Netzlisten zur FPGA-Spezifizierung von Abschnitten des Programms liegt. Solche Programme schaffen keine tatsächliche Hardware-Rekonfigurierung, sondern sorgen lediglich für die Fähigkeit zur Abbildung auf ein anderes Teil der Hardware.

In gleicher Weise verwenden einige Systeme einen Standard-Mikroprozessor mit einigen konfigurierbaren Logik-Betriebsmitteln. Diese Betriebsmittel (Resourcen) werden verwendet, um spezielle Befehle bzw. Instruktionen zu realisieren, die die Ausführung von bestimmten Programmen beschleunigen. Siehe beispielsweise R. Razdan und M.D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", in Proceedings of the Twenty-Seventh Annual Microprogramming Workshop, IEEE Computer Society Press, 1994. Solche Systeme werden typischerweise als Zentralprozessoreinheit (CPU) realisiert, und zwar mit einem Abschnitt des Silizium-Chips, der verwendet wird, um einen FPGA zu realisieren. Die CPU besitzt einen festen bzw. fixierten Datenpfad, mit dem die FPGAs verbunden sind. Der Kompiler kombiniert ausgewählte Assembler-Codesequenzen in Einzelbefehlanweisungen zur Ausführung durch ein FPGA. Jedoch arbeiten solche Systeme generell nur auf Ebenen eines bestehenden Assembler-Sprachcodes und erfordern eine angrenzende, feste Befehlsatzarchitektur (nachfolgend als ISA oder auch IS-Architektur bezeichnet) als Ausgangspunkt. Außerdem halten solche Systeme generell keine Laufzeit-Rekonfigurierung bereit. Schließlich sind solche Systeme nicht weit anwendbar und sorgen üblicher Weise nicht für eine deutliche Geschwindigkeitsverbesserung im Vergleich zu anderen herkömmlichen Systemen.

Obwohl die zuvor genannten Systeme jeweils ein gewisses Niveau für die Rekonfigurierbarkeit von Hardware schaffen, beschreibt keines von diesen ein Verfahren oder eine Vorrichtung zur Zusammenfassung von binären Maschinensprache-Befehlen und von Daten gemeinsam mit den Hardware-Konfigurierungen, die notwendig sind, um die Maschinenbefehle in der in diesem Patent beanspruchten Art und Weise auszuführen. Außerdem offenbaren die bekannten Systeme weder eine Mehrfacharchitektur-ISA-Rekonfiguration auf der Ebene der Granularität, die vergleichbar zu RISC- oder CISC-Befehlen ist, die hierin beansprucht werden, noch Kompilierungsverfahren innerhalb der C-Sprachensyntax zum Ausführen auf dynamisch rekonfigurierten IS-Architekturen, wie diese hierin beansprucht werden.

Erfindungsgemäß wird ein System und ein Verfahren zur Kompilierung von Quellcode geschaffen, der beispielsweise in C oder Pascal geschrieben ist, um ausführbare Dateien bzw. Programme zur Verwendung in einer dynamisch rekonfigurierbaren Prozessoreinheit zu erzeugen, die eine wahlweise veränderbare interne Hardwareorganisation aufweist. Bei einer Ausführungsform können erfindungsgemäß Maschinenbefehle und Daten gemeinsam mit Hardware-Konfigurationen zusammengefaßt werden, welche erforderlich sind, um die Maschinenbefehle auszuführen. In der rekonfigurierbaren Architektur besteht jeder einzelne Prozessor beispielsweise aus: Einer rekonfigurierbaren Prozessor-Hardware, wie beispielsweise einem vollständig FPGA-basierten Prozessor, einem Datenspeicher und einem Programmspeicher, einer Parallel-Verbindungseinheit und einem wiederbeschreibbaren Speicher für FPGA-Konfigurationsbits. Durch dynamisches Laden von FPGA-Konfigurationsbitströmen realisiert die vorliegende Erfindung einen dynamischen ISA-Computer, der eine hohe Leistungsfähigkeit erreicht, in dem ISAs verwendet werden, die für spezielle Phasen der Ausführung der Applikation optimiert sind.

Bei der erfindungsgemäßen Architektur werden Applikationen als Software zur Verfügung gestellt, wird Hardware in der Form von Schaltungen (ein zentrales Service-Modul, Prozessormodule, Eingabe/Ausgabe-Module (I/O)) sowie Bitströme für Befehlsatzarchitekturen (ISAs) bereitgestellt, die auf den FPGAs des Prozessormoduls beheimatet sind. Ein ISA bzw. eine IS-Architektur ist ein primitiver Satz von Befehlen, der dazu verwendet werden kann, einen Computer zu programmieren. Applikationssoftware wird mit Hilfe von FPGAs ausgeführt, die als ISAs auf den Prozessormodulen konfiguriert sind.

Die vorliegende Erfindung beschreibt ein System, das ausgelegt ist, so daß FPGA-Konfigurationsbitströme statisch während der Kompilierung mit dem Programm verbunden bzw. verknüpft werden können, das diese ausführt, und daß diese zum dynamischen Schalten von ISAs und/oder FPGA-Applikationselementrealisierungen unabhängig und in Echtzeit programmiert werden können.

Die ISAs führen Programmbefehle aus, die in RAM **133** gespeichert sind. Diese Programmbefehle umfassen wahlweise eine oder mehrere Rekonfigurations-Übersetzungsanweisungen (reconfiguration directives). Bei Auswahl einer Rekonfigurations-Übersetzungsanweisung bzw. Rekonfigurations-Betriebsanweisung wird die Hardware rekonfiguriert, um für eine optimierte Realisierung einer bestimmten Befehlsatzarchitektur zu sorgen. Zusätzlich zu ihrer spezifischen Funktionalität umfaßt jede ISA einen Befehl oder eine Übersetzungsanweisung, der bzw. die veranlaßt, daß eine andere ISA in den rekonfigurierbaren Prozessor geladen wird, so daß die Ausführung der Software anschließend unter Verwendung der neuen ISA fortfährt.

Weil die Speicherstelle der ISA-Bitströme im Speicher ein Argument für den Rekonfigurationsbefehl darstellt, wird diese Speicherstelle vorzugsweise zum Zeitpunkt der Verbindung (link) oder des Ladens bestimmt, was auch für die Speicherstellen für Funktions-Aufrufziele und -Variablen gilt. Ebenso wie für diese Funktionen und Variablen hat es sich als wünschenswert herausgestellt, symbolische Namen für die Adressen eines Bitstroms zu verwenden. Die vorliegende Erfindung verwendet ein Objektdatei-Format, das die Vorstellung einer ausführbaren Software auf ISA-Bitströme erweitert. Daraus resultieren einige Vorteile, wie beispielsweise:

- Tools bzw. Werkzeuge können leicht aufgebaut bzw. erstellt werden. Weil die Rekonfigurierung als ein Befehl und Bitströme als Daten behandelt wird, können standardmäßige Software-Verbindungsverfahren eingesetzt werden, um softwaregesteuerte Hardwareänderungen mit den erforderlichen Bitstrom zu binden. Keine neue Softwaretechnologie über die Bitstrom-als-Daten-Abstraktion muß erzeugt werden.
- Flexibilität beim Laden. Indem Rekonfigurationsdaten auf einen Teil des Ausführbaren isoliert werden, wird die Fähigkeit Konfigurationen in geschützte Bereiche des Speichers zu laden, vereinfacht. Mit Gesichtspunkten der Speicherausrichtung wird man auf strukturierte Weise leicht fertig, wie nachfolgend ausführlicher beschrieben werden wird.
- Das Laden wird vereinfacht. Alle Daten, die erforderlich sind, um das Programm auszuführen, werden in einer einzigen Datei aufbewahrt, so daß keine Ladezeit-Identifikation und keine Lokalisierung von Bitströmen ausgeführt zu werden braucht, falls das Ausführbare bzw. das Programm statisch verbunden wird.
- Das Konfigurationsmanagement wird vereinfacht. Nur eine einzige Datei braucht zur gleichen Zeit beibehalten werden, sobald ein Programm gebunden worden ist. Dies vereinfacht die Vorgehensweise zur Verteilung von Applikationen auf einzelne Geräte und entfernte Stellen.

Nachfolgend wird die Erfindung in beispielhafter Weise und unter Bezugnahme auf die Zeichnungen beschrieben, in denen:

Fig. 1 ein Blockschema der Hardwarekomponenten einer dynamisch rekonfigurierbaren Rechenarchitektur ist;

Fig. 1A ein Blockschema eines erfindungsgemäßen Prozessormoduls ist;

Fig. 1B und 1C Blockschema einer Systemarchitektur zur Realisierung der Erfindung sind, die ein Beispiel für die Rekonfiguration eines FPGAs zeigen;

Fig. 2 ein Beispiel für ein Programmlisting ist, das Rekonfigurations-Übersetzungsanweisungen enthält;

Fig. 3 ein Flußdiagramm für ein Gesamt-Kompilierverfahren ist, das von einem Kompiler bzw. Übersetzungsprogramm zum dynamisch rekonfigurierbaren Rechnen ausgeführt wird;

Fig. 3A und 3B ein Flußdiagramm von bevorzugten Kompiliervorgängen sind, die von einem Kompiler zum dynamisch rekonfigurierbaren Rechnen ausgeführt werden;

Fig. 3C ein Flußdiagramm von weiteren Kompiliervorgängen ist, die von einem Kompiler zum dynamisch rekonfigurierbaren Rechnen ausgeführt werden;

Fig. 4 ein Blockschema eines Kompiliersystems gemäß der vorliegenden Erfindung ist;

Fig. 5 ein Schema eines Objektdateiformats aus dem Stand der Technik ist;

Fig. 6 ein Flußdiagramm für ein Verfahren zum Erhalten eines Programmzustands gemäß der vorliegenden Erfindung ist;

Fig. 7 ein Flußdiagramm für ein Verfahren zur strukturierten Rekonfiguration gemäß der vorliegenden ist; und

Fig. 8A, 8B und 8C Diagramme von Stapelinhalten während einer strukturierten Rekonfiguration gemäß der vorliegenden Erfindung darstellen.

Fig. 9 ein Blockdiagramm einer bevorzugten Ausführungsform eines Systems für ein skalierbares, paralleles, dynamisch rekonfigurierbares Berechnen, gemäß der Erfindung;

Fig. 10 ein Blockdiagramm einer bevorzugten Ausführungsform einer S-Einrichtung gemäß der Erfindung;

Fig. 11A ein beispielhaftes Programmauflisten, das Rekonfigurationsanweisungen enthält;

Fig. 11B ein Ablaufdiagramm von herkömmlichen Kompilieroperationen, die während der Kompilation einer Folge von Programmbefehlen durchgeführt worden sind;

Fig. 11C und 11D ein Ablaufdiagramm von bevorzugten Kompilieroperationen, welche mittels eines Kompilierers für ein dynamisch rekonfigurierbares Berechnen durchgeführt worden sind;

Fig. 12 ein Blockdiagramm einer bevorzugten Ausführungsform einer dynamisch rekonfigurierbaren Verarbeitungseinheit der Erfindung;

Fig. 13 ein Blockdiagramm einer bevorzugten Ausführungsform einer Befehlsabrufeinheit der Erfindung;

Fig. 14 ein Zustandsdiagramm, das eine bevorzugte Gruppe von Zuständen zeigt, die durch eine Befehlszustands-Ablaufsteuereinheit der Erfindung gestützt sind;

Fig. 15 ein Zustandsdiagramm, das eine bevorzugte Gruppe von Zuständen zeigt, welche durch Unterbrechungslogik der Erfindung getragen sind;

Fig. 16 ein Blockdiagramm einer bevorzugten Ausführungsform einer Datenoperationseinheit der Erfindung;

Fig. 17A ein Blockdiagramm einer ersten beispielhaften Ausführungsform der Datenoperationseinheit, welche durch die Ausführung einer universellen Außenschleifen-Befehlssatz-Architektur konfiguriert ist;

Fig. 17B ein Blockdiagramm einer zweiten beispielhaften Ausführungsform der Datenoperationseinheit, welche für die Verwirklichung einer Innenschleifen-Befehlssatz-Architektur konfiguriert ist;

Fig. 18 ein Blockdiagramm einer bevorzugten Ausführungsform einer Adressenoperationseinheit der Erfindung;

Fig. 19A ein Blockdiagramm einer ersten beispielhaften Ausführungsform der Adressenoperationseinheit, welche für die Verwirklichung einer universellen Außenschleifen-Befehlssatz-Architektur konfiguriert ist;

Fig. 19B ein Blockdiagramm einer zweiten beispielhaften Ausführungsform der Adressenoperationseinheit, welche für die Verwirklichung einer Innenschleifen-Befehlssatz-Architektur konfiguriert ist;

Fig. 20A ein Diagramm, das eine beispielhafte Zuordnung von rekonfigurierbaren Hardware-Ressourcen zwischen

der Befehlsabrufeinheit, der Datenoperationseinheit und der Adressenoperationseinheit für eine Außenschleifen-Befehlssatz-Architektur zeigt;

Fig. 20B ein Diagramm einer beispielhaften Zuordnung von rekonfigurierbaren Hardware-Ressourcen zwischen der Befehlsabfrageeinheit der Datenoperationseinheit und der Adressenoperationseinheit für eine Innenschleifen-Befehlssatz-Architektur;

Fig. 21 ein Blockdiagramm einer bevorzugten Ausführungsform einer T-Einrichtung der Erfindung;

Fig. 22 ein Blockdiagramm einer Verbindungs-Ein-/Ausgabeeinheit der Erfindung;

Fig. 23 ein Blockdiagramm einer bevorzugten Ausführungsform einer Ein-/Ausgabe-T-Einrichtung der Erfindung;

Fig. 24 ein Blockdiagramm einer bevorzugten Ausführungsform einer universellen Verbindungsmatrix der Erfindung, und

Fig. 25A und 25B ein Ablaufdiagramm eines bevorzugten Verfahrens für ein skalierbares, paralleles, dynamisch rekonfigurierbares Berechnen gemäß der Erfindung.

Die vorliegende Erfindung ist auf ein Kompiliersystem und ein Verfahren zur Erzeugung ausführbarer Dateien zur Verwendung bei einer dynamisch rekonfigurierbaren Prozessoreinheit gerichtet, deren Hardware-Konfiguration nachfolgend insbesondere anhand der **Fig.** 9 bis 25B beschrieben wird.

In **Fig.** 1 ist ein Blockschema eines skalierbaren, parallelen, dynamisch rekonfigurierbaren Computers **10** zum Ausführen von Objektdateien gezeigt, die gemäß der vorliegenden Erfindung erzeugt wurden. Der Computer **10** umfaßt vorzugsweise mindestens eine S-Einrichtung **12**, eine T-Einrichtung **14**, die jeder S-Einrichtung **12** entspricht, eine Mehrzweck-Verbindungsmatrix (General Purpose Interconnect Matrix; GPI-Matrix) **16**, mindestens eine Eingabe-/Ausgabe-T-Einrichtung **18**, eine oder mehrere Eingabe-/Ausgabeeinrichtungen **20** und eine Master-Zeitbasiseinheit **22**. Bei der bevorzugten Ausführungsform umfaßt der Computer **10** mehrere S-Einrichtungen **12** und somit auch mehrere T-Einrichtungen **14** und außerdem mehrere Eingabe-/Ausgabe-T-Einrichtungen **18** und mehrere Eingabe-/Ausgabeeinrichtungen **20**.

Jede der S-Einrichtungen **12**, T-Einrichtungen **14** und Eingabe-/Ausgabe-T-Einrichtungen **18** hat einen Mastertakteingang, der mit dem Taktausgang der Masterzeitbasiseinheit **22** verbunden ist. Jede S-Einrichtung **12** hat einen Eingang und einen Ausgang, der mit ihrer entsprechenden T-Einrichtung **14** verbunden ist. Zusätzlich zu dem Eingang und dem Ausgang, die mit der entsprechenden S-Einrichtung **12** verbunden sind, weist jede T-Einrichtung **14** einen Wegsteuerungseingang (routing input) und einen Wegsteuerungsausgang auf, die mit der GPI-Matrix **16** verbunden sind. Dementsprechend hat jede Eingabe-/Ausgabe-T-Einrichtung **18** einen Eingang und einen Ausgang, die mit einer Eingabe-/Ausgabeeinrichtung **20** verbunden sind, sowie einen Wegsteuerungseingang und einen Wegsteuerungsausgang, der mit der GPI-Matrix **16** verbunden ist.

Bei jeder S-Einrichtung **12** handelt es sich um einen dynamisch rekonfigurierbaren Rechner. Die GPI-Matrix **16** stellt ein paralleles Punkt-zu-Punkt-Verbindungsmittel bzw. ein paralleles Maschen-Verbindungsmittel dar, das die Kommunikation zwischen den T-Einrichtungen **14** erleichtert. Der Satz von T-Einrichtungen **14** und die GPI-Matrix **16** bilden ein paralleles Punkt-zu-Punkt-Verbindungsmittel für einen Datentransfer zwischen Speichern, die der S-Einrichtung **12** zugeordnet sind. In ähnlicher Weise bilden die GPI-Matrix **16**, der Satz von T-Einrichtungen **14** und der Satz von Eingabe-/Ausgabe-T-Einrichtungen **18** ein paralleles Punkt-zu-Punkt-Verbindungsmittel für einen Eingabe-/Ausgabetransfer zwischen S-Einrichtungen **12** und jeder Eingabe-/Ausgabeeinrichtung **20**. Die Master-Zeitbasiseinheit **22** umfaßt einen Oszillator, der jeder S-Einrichtung **12** und jeder T-Einrichtung **14** ein Master-Taktsignal zur Verfügung stellt.

In einer beispielhaften Ausführungsform ist jede S-Einrichtung **12** durch Verwendung eines Xilinx XC4013 (Xilinx, Inc., San Jose, CA) feldprogrammierbaren Gate-Arrays (FPGA) bzw. Logikanordnung ausgeführt, das mit einem 64 MB Direktzugriffsspeicher (RAM) verbunden ist. Jede T-Einrichtung **14** ist durch Verwendung von annähernd 50% der rekonfigurierbaren Hardware-Betriebsmittel in einem Xilinx XC4013 FPGA ausgeführt, ebenso jede Eingabe-/Ausgabe-T-Einrichtung **18**. Die GPI-Matrix **16** ist als ein ringförmiges Verbindungsmaschennetz ausgeführt. Die Master-Zeitbasiseinheit **22** ist ein Taktoszillator, der mit einer Taktverteilungsschaltung verbunden ist, um für eine systemweite Frequenzreferenz zu sorgen, wie nachfolgend anhand der **Fig.** 9 bis 25B beschrieben wird. Vorzugsweise übertragen die GPI-Matrix **16**, die T-Einrichtungen **14** und die Eingabe-/Ausgabe-T-Einrichtungen **18** Information entsprechend dem Punkt-zu-Punkt-Protokoll des ANSI/IEEE-Standard 1596–1992, wodurch ein skalierbares kohärentes Interface (SCI) definiert ist.

In **Fig.** 1A ist ein Blockschema eines Prozessormoduls **130** gezeigt, das in einer Ausführungsform der vorliegenden Erfindung verwendet wird. Der S-Einrichtungs-FPGA **12** ist mit einem zugeordneten bzw. reservierten Bitstromspeicher **132** und einem Programm-/Datenspeicher **133**, einer oder mehreren T-Einrichtungen **14** und einer Takterzeugungsschaltung verbunden, wie beispielsweise einen Taktgenerator **131**, um ein Prozessormodul **130** zu bilden. Das Modul **130** ist mit anderen, vergleichbaren Modulen über die T-Einrichtungen **14** in einer solchen Art und Weise verbunden, die einen Parallelbetrieb erleichtert. Der Programm-/Datenspeicher **133** speichert Programmbefehle und ist in Form eines üblichen RAMs realisiert. Der Bitstromspeicher **132** speichert Bitströme, die die FPGA-Konfigurationen beschreiben. In einer Ausführungsform ist der Programm-/Datenspeicher **133** als dynamisches RAM (DRAM) implementiert und der Bitstromspeicher **132** als statisches RAM (SRAM).

In den **Fig.** 1B und 1C sind Beispiele für eine FPGA-Rekonfiguration gezeigt, um ISAs in rekonfigurierbarer Architektur zu realisieren. Die Figuren zeigen Blockschema einer Systemarchitektur zur Realisierung der vorliegenden Erfindung, wobei die S-Einrichtungs-FPGA **12** umprogrammiert ist, so daß sie eine arithmetische Logikeinheit (ALU) **143** in **Fig.** 1B umfaßt sowie einen finiten Impulsantwortfilter (FIR) **148** in **Fig.** 1C. Ein Bitstrom-RAM **132** und ein Programm-/Daten-RAM **133** ist vorgesehen. Der Speicherbus **149** hält einen Kommunikationskanal zwischen dem S-Einrichtungs-FPGA **12** und RAM **132** und **133** bereit. Die FPGA-Konfigurationshardware **140** ermöglicht die Rekonfiguration des S-Einrichtungs-FPGAs **12** entsprechend den ISA-Bitströmen vom Bitstrom-RAM **132**. Konfigurationen des S-Einrichtungs-FPGA **12** umfassen beispielsweise Datenregister bzw. Datenspeicher **141**, Adreßregister **142**, einen Registermultiplexer **144** und ein Speicherdatenregister **145**. Jede oder alle dieser Komponenten kann modifiziert oder in anderen Konfigurationen entfernt werden, was von dem Bitstrom abhängt. Beispielsweise taucht Alu **143** in der in **Fig.** 1B gezeigten Konfiguration auf, ist aber in der Konfiguration aus **Fig.** 1C durch den FIR-Filter **148** ersetzt.

Vorzugsweise speichert der Computer **10** Programmbefehle in RAM wahlweise, und zwar einschließlich von Rekonfigurations-Übersetzungsanweisungen zur Rekonfigurierung von Computer **10**, indem die Konfiguration der S-Einrichtung **12** geändert wird. In **Fig. 2** ist ein beispielhaftes Programmlisting **50** gezeigt, das einen Satz von Außenschleifenabschnitten **52**, einen ersten Innenschleifenabschnitt **54**, einen zweiten Innenschleifenabschnitt **55**, einen dritten Innenschleifenabschnitt **56**, einen vierten Innenschleifenabschnitt **57** und einen fünften Innenschleifenabschnitt **58** umfaßt. Wie der Fachmann weiß, verweist der Begriff "Innenschleife" auf einen iterativen Abschnitt eines Programms, der dafür verantwortlich ist, einen ganz bestimmten Satz von verwandten Operationen auszuführen; und der Begriff "Außenschleife" verweist auf die Abschnitte eines Programms, die hauptsächlich dafür verantwortlich sind, Mehrzweck-Operationen bzw. universelle Operationen und/oder eine Übertragungssteuerung von einem Innenschleifenabschnitt zum anderen durchzuführen. Im allgemeinen führen die Innenschleifenabschnitte **54** bis **58** eines Programms spezifische Operationen an möglicherweise großen Datensätzen durch. Eine oder mehrere der Rekonfigurations-Übersetzungsanweisungen kann einem vorgegebenen Innenschleifenabschnitt **54**, **55**, **56**, **57** oder **58** zugeordnet sein, so daß sich eine geeignete ISA im Kontext befinden wird, wenn der Innenschleifenabschnitt ausgeführt wird. Im allgemeinen werden für ein beliebiges vorgegebenes Programm die Außenschleifenabschnitte **52** des Programmlistings **50** eine Vielzahl von Mehrzweck-Befehlsarten umfassen, während die Innenschleifenabschnitte **54**, **56** des Programmlistings **50** aus vergleichsweise wenig Befehlsarten bestehen werden, die dazu verwendet werden, einen spezifischen Satz von Operationen auszuführen.

In einer beispielhaften Programmauflistung **50** erscheint eine erste Rekonfigurations-Übersetzungsanweisung zu Beginn des ersten Innenschleifenabschnitts **54** und erscheint eine zweite Rekonfigurations-Übersetzungsanweisung am Ende des ersten Innenschleifenabschnitts **54**. Dementsprechend erscheint eine dritte Rekonfigurations-Übersetzungsanweisung zu Beginn des zweiten Innenschleifenabschnitts **55**; eine vierte Rekonfigurations-Übersetzungsanweisung erscheint zu Beginn des dritten Innenschleifenabschnitts **56** usw. Jeder Rekonfigurationsbefehl verweist vorzugsweise auf einen Konfigurationsdatensatz, der von einem Bitstrom dargestellt wird. Der Bitstrom spezifiziert eine interne Hardware-Organisation für jede S-Einrichtung **12**, und zwar einschließlich einer dynamisch rekonfigurierbaren Prozessoreinheit (nachfolgend DRPU genannt), einer Adreß-Betriebseinheit (AOU), einer Befehl-Abrufeinheit (IFU) und einer Datenbetriebseinheit (DOU) (nicht gezeigt). Eine solche Hardware-Organisation ist gedacht und optimiert zur Realisierung einer bestimmten Befehlsatzarchitektur (Instruction Set Architecture; ISA). Eine IS-Architektur ist ein einfacher Satz oder Kernsatz von Befehlen bzw. Instruktionen, die dazu verwendet werden können, um einen Rechner zu programmieren. Eine IS-Architektur definiert Befehlsformate, Operationscodes, Datenformate, Adressiermodes, Ausführungs-Steuerflags und programmzugängliche Register bzw. Verzeichnisse. Bei der rekonfigurierbaren Rechnerarchitektur, die zur Ausführung von Objektdateien eingesetzt wird, die gemäß der vorliegenden Erfindung erzeugt werden, kann jede S-Einrichtung sehr rasch und in Echtzeit konfiguriert werden, um unmittelbar eine Folge von IS-Architekturen durch Verwendung eines eindeutigen Konfigurationsdatensatzes für jede gewünschte IS-Architektur zu realisieren, die durch einen Bitstrom spezifiziert wird. Somit wird jede IS-Architektur mit einer speziellen, internen Hardware-Organisation realisiert, wie sie durch einen entsprechenden Konfigurationsdatensatz spezifiziert wird. Folglich entsprechen in dem Beispiel aus **Fig. 2** die ersten fünf Innenschleifenabschnitte **54** bis **58** jeweils einer eindeutigen IS-Architektur 1, 2, 3, 4 bzw. k. Der Fachmann erkennt, daß jede nachfolgende IS-Architektur nicht eindeutig zu sein braucht. Folglich könnte ISA k 1, 2, 3, 4 oder irgendeine andere ISA sein. Der Satz von Außenschleifenabschnitten **52** entspricht ebenfalls einer eindeutigen ISA, nämlich ISA 0. Während der Programmausführung kann die Auswahl nachfolgender Rekonfigurations-Übersetzungsanweisungen von den Daten abhängen. Bei Auswahl einer gegebenen Rekonfigurations-Übersetzungsanweisung werden im Anschluß daran Befehle bzw. Anweisungen nach einer entsprechenden ISA über eine eindeutige S-Einrichtungs-Hardwarekonfiguration ausgeführt, wie sie durch den Bitstrom spezifiziert wird, auf den durch die Rekonfigurations-Übersetzungsanweisung verwiesen wird.

Mit der Ausnahme von Rekonfigurations-Übersetzungsanweisungen umfaßt das beispielhafte Programmlisting **50** aus **Fig. 2** übliche Hochsprachen-Anweisungen, beispielsweise Anweisungen, die entsprechend der C-Programmiersprache geschrieben sind.

Der Fachmann erkennt, daß der Einbau von einer oder mehreren Rekonfigurations-Übersetzungsanweisungen in einer Folge von Programmanweisungen einen Kompiler bzw. ein Übersetzungsprogramm erfordert, der bzw. das modifiziert wurde, um den Rekonfigurations-Übersetzungsanweisungen Rechnung zu tragen. Folglich umfaßt das erfindungsgemäße Kompiliersystem und das erfindungsgemäße Verfahren Vorgänge einschließlich von Rekonfigurations-Übersetzungsanweisungen durch Zusammenfassung von Verweisungen auf Bitströme, die Hardware-Konfigurationen beschreiben, und durch Übersetzung bzw. Kompilierung von Quellcode entsprechend den Spezifikationen von bestimmten ISAs, die durch die Rekonfigurations-Übersetzungsanweisungen identifiziert werden.

In einer Ausführungsform der vorliegenden Erfindung unterstützen alle dem Computer **10** zur Verfügung stehende ISAs die folgenden Vorgänge:

- Einen Stapelzeiger (stack pointer; SP) und ein Zeiger Adreßverzeichnis für nächste Befehle (Next Instruction Pointer Address Register; NIPAR; auch bekannt als Programmzähler (PC)), um einen stapel-basierten Speicher von Informationen und Parametern während der Rekonfiguration zu realisieren;
- geeignete Befehle in Assemblersprache zur Flußsteuerung, und zwar einschließlichbeispielsweise von "jsr" bzw. "jump to subroutine" für einen Unterprogramm-Einsprung und "rts" bzw. "return to subroutine" für eine Unterprogramm-Rückkehr; und
- eine geeignete Speicher-Schnittstelleneinheit zum Speichern und Laden von Verzeichniswerten in bzw. aus dem Stapel.

Die Betriebsweise dieser Komponenten zur Realisierung einer Rekonfiguration wird nachfolgend anhand der **Fig. 6** bis 8C beschrieben.

Komponenten des Kompiliersystems

In **Fig. 4** ist ein Blockschema eines erfindungsgemaßen Kompiliersystems dargestellt. Das Kompiliersystem und das erfindungsgemäße Verfahren läuft auf einer typischen Workstation oder einem PC, der ein übliches Betriebssystem, wie beispielsweise Unix, verwendet. Die Unix-Umgebung ist wegen der großen Verfügbarkeit von Quellcode für Software-Entwicklungstools und der Robustheit der Benutzer-Umgebung vorteilhaft. Wie der Fachmann erkennen wird, könnte das erfindungsgemäße System und das erfindungsgemäße Verfahren direkt auf einem rekonfigurierbaren Computer laufen. In **Fig. 3** ist ein Flußdiagramm für ein erfindungsgemäßes Gesamtverfahren zur Kompilierung bzw. Übersetzung, zur Assemblierung, zur Verbindung bzw. Verknüpfung und zum Laden gezeigt. Die Kompilierschritte aus **Fig. 3** werden nachfolgend anhand der **Fig. 3A** bis 3C ausführlicher beschrieben.

Die Quelldateien **401** werden mit Hilfe eines speziell modifizierten C-Kompilers **402** kompiliert, der nachfolgend beschrieben wird. Der Kompiler **402** liest **(301)** die Quelldateien **401**, die Quellcode-Befehlsanweisungen enthalten, von einem Plattenspeicher oder von einem anderen Eingabe- oder Speichergerät. Der Kompiler **402** identifiziert **(302)** dann eine ISA für einen Untersatz von Quellcodebefehlsanweisungen. In einer Ausführungsform werden ISAs von Rekonfigurations-Übersetzungsanweisungen identifiziert, wie nachfolgend ausführlicher beschrieben wird. Der Kompiler **402** erzeugt **(303)** geeignete Rekonfigurations-Übersetzungsanweisungen, um die identifizierte ISA zu spezifizieren, und kompiliert **(304)** den Untersatz von Befehlen zur Ausführung durch die identifizierte ISA, um Anweisungen in Assemblersprache zu erzeugen. Der Kompiler **402** bestimmt dann **(305)**, ob ein nachfolgender Untersatz von Befehlsanweisungen (typischerweise eine separate Funktion innerhalb der Quelldatei **401**) mit einer anderen ISA kompiliert werden soll. In einer Ausführungsform wird eine solche Bestimmung wiederum dadurch ausgeführt, daß die Rekonfigurationsübersetzungsanweisungen überprüft werden. Falls eine andere ISA identifiziert wird, kehrt der Kompiler **402** zu Schritt **302** zurück.

Anderenfalls, wenn das Ende der Quelldatei erreicht wird, werden die Assemblersprachenanweisungen vom Assembler **409** assembliert **(306)**, um Objektdateien **403** zu erzeugen. Die Objektdateien **403** werden mit Hilfe des Softwarelinkers bzw. Softwarebinders **404** verbunden **(307)**, der modifiziert wurde, um Bitstrom-Speicherstellen und abgeglichene bzw. synchronisierte 64-Bit-Adressen zu behandeln, um ein ausführbares Programm **405** zu erzeugen. Wie nachfolgend beschrieben wird, enthält das ausführbare Programm **405** aufgelöste Verweise (Referenzen) auf ISA-Bitströme **406**, die FPGA-Architekturen festlegen. Nachdem das ausführbare Programm **405** vom Binder **404** erzeugt wurde, wird dieses über die Netzwerkverbindung **408** an das Ladeprogramm **407** gesendet, das auf einem rekonfigurierbaren Computer **10** abläuft, zum Laden **(308)** in den Computer **10**. Für den Fall einer dynamischen Verbindung werden ISA-Bitströme **406** auch über die Netzwerkverbindung **408** an das Ladeprogramm **407** gesendet.

Beliebige und strukturierte Rekonfigurierung

In einer Ausführungsform läßt der Kompiler **402** eine beliebige Rekonfigurierung (arbitrary reconfiguration) zu, bei der die Rekonfigurations-Übersetzungsanweisungen an einer beliebigen Stelle in dem Quellcode lokalisiert sein können. Bei einer anderen Ausführungsform läßt der Kompiler **402** eine strukturierte Rekonfigurierung (structured reconfiguration) zu, bei der Rekonfigurierungs-Übersetzungsanweisungen nur zugelassen wird, wenn von einer Funktion aufgerufen oder zurückgekehrt wird, so daß jede Funktion mit einer einzelnen ISA bezeichnet wird, die während des gesamten Ablaufs der Funktion im Kontext bzw. Zusammenhang sein soll. Während eine beliebige Rekonfigurierung zusätzliche Flexibilität und einen kleineren Quellcode ermöglicht, sorgt eine strukturierte Rekonfigurierung für eine bessere Vorhersagbarkeit und einen besseren Determinismus beim Laden einer ISA, was zu einer größeren Zuverlässigkeit führt. Weil der Maschinencode generell für verschiedene ISAs verschieden ist, wird der Determinismus bevorzugt, so daß der Kompiler in der Lage ist, einen geeigneten Maschinencode für ein bestimmtes Segment des Quellcodes zu erzeugen. Die beliebige Rekonfigurierung kann zu nicht deterministischen Situationen führen, wenn diese mit gewissen Konditionalkonstrukten im Quellcode kombiniert wird. Diese Situationen werden durch Verwendung einer strukturierten Rekonfigurierung beseitigt.

Der folgende Auszug aus einem Code stellt ein Beispiel für eine nicht deterministische Rekonfiguration dar, die auftreten kann, wenn eine beliebige Rekonfiguration verwendet wird:

```
#pragma reconfig ISAO
...
x = O
if (a != O) {
#pragma reconfig ISA1
} else {
#pragma reconfig ISA2
}
y = x + 2;
...
```

Die ISA im Kontext nach der if-Anweisung kann zum Zeitpunkt der Kompilierung nicht bestimmt werden, weil es zur Laufzeit zwei mögliche Pfade für den Steuerfluß gibt, von denen jeder eine Rekonfiguration zu einer anderen ISA bewirkt. Deshalb kann der Kompiler für diese Prozedur keinen gültigen Maschinencode ausgeben, solange ISA1 und ISA2 binär kompatibel sind. Ein solcher Nichtdeterminismus wird beseitigt, wenn eine strukturierte Rekonfiguration verwen-

det wird, weil nur eine ISA pro Funktion spezifiziert werden kann.

Bei dem oben genannten Beispiel ist der Wert der Variablen x während des Rekonfigurationsvorgangs geschützt, so daß auf diesen von der neuen IS-Architektur zugegriffen werden kann. In einer Ausführungsform wird der Wert in einem Register bzw. Verzeichnis von ISA0 in herkömmlicher Art und Weise abgespeichert. Die Rekonfigurierung in ISA1 oder ISA2 kann jedoch bewirken, daß dieses Verzeichnis aufhört, zu existieren oder seinen Wert verliert, so daß man sich nicht auf das Verzeichnis verlassen kann, wenn es den Wert von x nach der Rekonfiguration liefert. Der Kompiler **402** überwacht deshalb die lebenden Verzeichniswerte, die nach einer Rekonfiguration verwendet werden, um sicherzustellen, daß ihre Werte zur Verfügung stehen, wenn sie benötigt werden.

Wenn eine beliebige Rekonfiguration verwendet wird, legt der Kompiler **402** fest, wie eine darauffolgende ISA mit einem Zugriff auf eine Variable versehen wird, indem dieser berücksichtigt, wie die augenblickliche ISA die Variable abgespeichert hat, ebenso wie die Einrichtungen, auf die die nachfolgende ISA zugreifen muß. In der strukturierten Rekonfiguration wird der Stapel dazu verwendet, um Werte abzuspeichern, wie dies üblich ist, wenn Werte an eine aufgerufene Funktion übergeben oder von dieser abgerufen werden. Weil sich die Verzeichnisarchitektur während der Rekonfiguration radikal ändern kann, werden lebende Variablen von der scheidenden ISA abgespeichert und dann wieder von der nachfolgenden ISA geladen, wie nachfolgend ausführlicher im Zusammenhang mit **Fig. 6** erörtert wird.

Bei einer Ausführungsform realisiert der Kompiler **402** eine bekannte "Linear"-Optimierung (inlining optimization), um eine strukturierte Rekonfiguration zu ermöglichen, um den Aufwand bzw. Systemverwaltungsaufwand der JSR-Anweisung zu vermeiden. Inlining ist ein bekanntes Verfahren zur Optimierung der Kompilierung von Funktionsaufrufen, indem die Vorgänge einer aufgerufenen Funktion "in der Linie" aufgerufen werden, um so den Aufwand zu vermeiden, der mit dem Aufruf der Funktion in üblicher Weise verbunden ist. Somit kann ein Code-Segment, wie beispielsweise:

```
#pragma reconfig ISA1
jsr SUBROUTINE_A
#pragma reconfig ISA0
#pragma reconfig ISA2
jsr SUBROUTINE_B
#pragma reconfig ISA0
```

übersetzt werden durch:

```
#pragma reconfig ISA1
< code von SUBROUTINE_A >
#pragma reconfig ISA0
#pragma reconfig ISA2
jsr SUBROUTINE_B
#pragma reconfig ISA0
```

wodurch die Leistungsfähigkeit verbessert wird, indem die Notwendigkeit einer Sprunganweisung und einer Programmrückkehr umgangen wird und auch die zugeordneten Stapel-Schreibvorgänge, die beim Aufruf einer Funktion und bei der Rückkehr von der Funktion involviert sind.

Außerdem kann eine zusätzliche Optimierung erfolgen, indem alle Rekonfigurations-Übersetzungsanweisungen bis auf die letzte eliminiert werden, wenn mehr als eine Rekonfigurations-Übersetzungsanweisung in Folge auftritt. Somit kann die dritte Zeile (#pragma reconfig ISA0) von dem oben genannten Code-Segment gelöscht werden.

Wenn man eine beliebige Rekonfiguration verwendet, kann eine aufgerufene Funktion eine Rekonfiguration bewirken, die wirksam bleibt, auch nachdem der Programmfluß zu der aufgerufenen Funktion zurückkehrt. Beispielsweise beginnt eine aufgerufene Funktion, die die oben genannte Code-Auflistung enthält, in einer ISA, rekonfiguriert zweimal und kehrt dann zu einer aufrufenden Funktion zurück. Von nachfolgenden Anweisungen in der aufrufenden Funktion muß deshalb angenommen werden, daß diese die ISA von der letzten Rekonfiguration verwenden. In einer Ausführungsform führt der Kompiler **402** eine interprozedurale Analyse durch, um zu bestimmen, welche ISAs sich bei jedem Funktionsaufruf und bei jeder Funktionsrückkehr im Kontext befinden. Dort, wo Quelldateien separat in Objektdateien kompiliert werden, bevor diese in ausführbare Programmanweisungen gebunden werden, kann es schwierig oder unmöglich sein, zu bestimmen, welche ISA sich im Kontext befinden wird, nachdem eine aufgerufene Funktion zurückkehrt. In solchen Situationen kann die ISA-Information abgespeichert werden, beispielsweise in Header-Dateien bzw. Kopfinformationsdateien, um zu spezifizieren, welche ISA sich im Kontext bei einer Funktions-Einsprungstelle sowie bei einer Funktions-Austrittsstelle befindet, und zwar für alle externen Funktionen, die von einem Modul aufgerufen werden. Alternativ können Parameter unter Funktionen weitergeleitet werden, um die ISA-Kontexte zu spezifizieren.

Wenn eine strukturierte Rekonfiguration verwendet wird, ist die ISA-Information im Vereinbarungsteil der Funktion vorgesehen, so daß keine Notwendigkeit besteht, daß der Kompiler **402** ISA-Spezifikationen gegen den Steuerfluß verifiziert, und es gibt keine Möglichkeit, daß eine unerwartete Rekonfiguration während einer aufgerufenen Funktion auftritt.

Ein weiterer Vorteil einer strukturierten Rekonfiguration besteht darin, daß diese den Rekonfigurationsvorgang vom semantischen Standpunkt her besser wiedergibt. Weil eine Rekonfiguration generell ein Maß an Aufwand mit sich bringt, das zumindest vergleichbar mit dem Aufwand für einen Funktionsaufruf ist, und weil eine Rekonfiguration viele derselben Arten von Operationen beinhaltet, wie beispielsweise die Abspeicherung von Werten auf einem Stapel, ist es wün-

schenswert, eine ähnliche Syntax sowohl für die Rekonfiguration als auch für Funktionsaufrufe zu schaffen. Die strukturierte Rekonfiguration verbindet die Idee einer Rekonfiguration mit der Idee von Funktionsaufrufen und verwirklicht deshalb dieses semantische Ziel. Weitere Beispiele für eine strukturierte und beliebige Rekonfiguration werden nachfolgend erörtert.

Rekonfigurations-Übersetzungsanweisungen

Bei der bevorzugten Ausführungsform steht vir Rekonfigurations-Übersetzungsanweisungen #pragma, eine normale Meta-Syntax, die bei der C-Sprache vorgesehen ist, um Information an den Kompiler weiterzuleiten, die aus der Sprachsyntax herausfällt. Die Verwendung der #pragma-Syntax ermöglicht es, daß die Rekonfigurations-Übersetzungsanweisung im Kontext eines C-Programms verwendet werden kann. Ein Beispiel für eine Rekonfigurations-Übersetzungsanweisung, die man in dem Quellcode finden kann, würde wie folgt lauten:

#pragma func_isa func2 isa2.

Bei einer Ausführungsform sind drei #pragma-Übersetzungsanweisungen vorgesehen. Jede Übersetzungsanweisungen wird auf einem anderen Niveau der Granularität oder des Gültigkeitsbereichs betrieben und beeinflußt deshalb einen spezifischen Teil des Codes:

- reconfig: beeinflußt eine Zwischen-Rekonfiguration zu einer neuen ISA (Gültigkeitsbereich ist ein beliebiger Block des Codes);
- func_isa: spezifiziert für eine bestimmte Funktion eine ISA (Gültigkeitsbereich ist die Funktion); und
- default_func_isa: spezifiziert eine Standard-ISA (Gültigkeitsbereich ist die gesamte Datei).

Diese Rekonfigurations-Übersetzungsanweisungen resultieren in Registertransferniveau-Rekonfigurationsanweisungen (RTL), die den Kompiler mit Information versorgen, um zu bestimmen, welche ISA für jeden Block des Codes benötigt wird, wie nachfolgend ausführlicher beschrieben wird.

Das nachfolgende Codelisting stellt ein Beispiel für die Verwendung von jeder der oben genannten Übersetzungsanweisungen in einer strukturierten Rekonfigurationsumgebung dar.

```
1   #include "icarus_types.h"
2   #include "icarus_isas.h"
3   #include "fixed.h"
4   #pragma default_func_isa ISA0
5   uns8 color_map[256];

6   #pragma func_isa build_color_map FIXED_POINT_ISA
7   void
8   build_color_map(int16 contrast)
9   {
10      unsigned color;
11      uns8 *color_map_tmp = color_map;

12      for (color = 0; color < 255U; color++) {
13          color_map_tmp[color] =
                fixed_mul_int8(contrast, color);

14      }
15  }

16  void
17  map_contrast(int x, int y, uns8 *image)
18  {
19      register int i, tmp;

20      tmp = x * y;

21  #pragma reconfig BYTE_MAP_ISA
22      {
23          register int i;
24          register uns8 *map;
25          register uns8 *image_tmp;
26  #pragma isa_pragma map_pointer map
27  #pragma isa_pragma image_pointer image_tmp
28  #pragma isa_pragma loop_counter i

29          image_tmp = image;
30          map = color_map;
31          for (i = tmp; i>0; i--) {
32              *image_tmp = map[*image_tmp];
33              image_tmp++;
34          }
35      }
36  }

37  do_contrast(int x, int y, uns8 *image,
        uns8 contrast)
38  {
39      build_color_map(contrast);
40      map_contrast(x, y, image);
41  }
```

Zeile 4 des Codelistings stellt ein Beispiel für die default_func_isa-Übersetzungsanweisung dar, die spezifiziert, daß ISA0 für jegliche Funktionen verwendet werden soll, die keine andere ISA spezifizieren. Der Gültigkeitsbereich dieser Übersetzungsanweisung ist die gesamte Datei; deshalb gilt die Übersetzungsanweisung für das gesamte gezeigte Listing.

Zeile 6 des Codelistings stellt ein Beispiel für die func_isa-Übersetzungsanweisung, die spezifiziert, daß FIXED_POINT_ISA die geeignete ISA für die Funktion darstellt, die build_color_map genannt wird. Der Gültigkeitsbereich dieser Übersetzungsanweisung ist die spezifizierte Funktion.

Zeile 21 des Codelistings stellt ein Beispiel für die reconfig-Übersetzungsanweisung dar, die spezifiziert, daß BYTE_MAP_ISA die geeignete ISA für den Codeblock darstellt, der unmittelbar der Übersetzungsanweisung folgt. Der Gültigkeitsbereich dieser Übersetzungsanweisung ist der in den Zeilen 22 bis 35 des Codelistings gezeigte Codeblock.

Das folgende Codelisting stellt ein Beispiel für die Verwendung von jeder der oben genannten Übersetzungsanweisun-

gen in einer beliebigen Rekonfigurationsumgebung dar.

```
1     unsigned char color_map[256];

2     /* int is 16 bits for supercomputer 1A */
3     typedef unsigned int uns16;
4     typedef unsigned char uns8;

5     void
6     build_color_map(uns16 contrast)
7     {
8           int color;

9     #pragma reconfig FixedIsa
10          for (color = 0; color < 256; color++) {
11                color_map[color] =
                        fixed_mul_int8(contrast, color);
12          }
13    }

14    map_contrast(int x, int y, uns8 *image)
15    {
16          register int i, tmp;
17          register uns8 *map;

18          tmp = x * y;
19    #pragma reconfig ByteMapIsa
20    #pragma map_pointer map
21    #pragma map_counter i
22    #pragma target_pointer image
23          i = tmp;
24          map = color_map;

25          while (i--) {
26                *image = map[*image];
27                image++;
28          }
29    }

30    /* Beim Einsprung in do_contrast IsaO annehmen */
31    do_contrast(int x, int y, uns8 *image,
            BinFrac *contrast)
32    {
33          build_color_map(contrast);

34          map_contrast(x, y, image);
35    }
```

Zeilen 9 und 19 enthalten reconfig-Übersetzungsanweisungen, die solange wirksam bleiben, bis eine andere Rekonfigurations-Übersetzungsanweisung festgestellt wird. Für eine beliebige Rekonfiguration können die Übersetzungsanweisungen an einem beliebigen Punkt in dem Code auftreten und sie sind nicht auf die Funktionsebenen-Granularität begrenzt.

Kompilierungsverfahren

In den **Fig.** 3A und 3B ist ein Flußdiagramm eines bevorzugten Kompilierungsverfahrens gemäß der vorliegenden Erfindung gezeigt. **Fig.** 3A zeigt die Schritte, die von der Kompiler-Oberfläche ausgeführt werden, während **Fig.** 3B die Schritte zeigt, die vom Kompiler-Kern ausgeführt werden. Die Oberfläche interpretiert Rekonfigurations-Übersetzungsanweisungen und erzeugt RTL-Anweisungen, die vom Kern in üblicher Weise interpretiert werden können. Wie man weiß, sind RTL-Anweisungen ISA-unabhängige Zwischenniveauanweisungen, die den herkömmlichen Kompilern eingesetzt werden, die beispielsweise in dem GNU C-Kompiler (GCC), der von der Firma Free Software Foundation (Cambridge, MA) hergestellt wird. RTL kann entsprechend der Spezifikation des Stanford University Intermediate Format

(SUIF) ausgeführt werden, wie diese in Stanford SUIF Compiler Group, SUIF: A Parallelizing & Optimizing Research Compiler, Tech. Rep. CSL-TR-94-620, Computer Systems Lab, Stanford University, May 1994 offenbart ist. Beispielsweise könnte die Quellcode-Anweisung:

```
x = y + 3;
```

in RTL wie folgt dargestellt werden:

```
r1 <- y
r0 <- r1 + 3
x <- r0.
```

Das Verfahren aus den **Fig.** 3A und 3B zieht als Eingangsgröße die Quelldatei **401** heran, die eine Folge von Hochsprachen-Quellcodebefehlsanweisungen enthält und die auch mindestens eine Rekonfigurations-Übersetzungsanweisung enthält, die eine ISA zur Ausführung von nachfolgenden Anweisungen spezifiziert. Um dies zu erläutern, sei eine strukturierte Rekonfigurationsumgebung angenommen, bei der eine Rekonfiguration Funktion für Funktion erfolgt. Die Oberfläche des Kompilers **402** wählt (**600**) die nächste Hochsprachenanweisung von der Quelldatei **401** aus und stellt fest (**601**), ob die ausgewählte Hochsprachenanweisung ein Funktionsaufruf ist. Falls dies nicht der Fall ist, sendet (**603**) der Kompiler **402** einen RTL-Code für diese Anweisung.

Falls der Kompiler **402** in Schritt **601** feststellt, daß die Anweisung ein Funktionsaufruf ist, stellt der Kompiler **402** in Schritt **602** fest, ob die gerade aufgerufene Funktion in einer anderen ISA als der gerade im Kontext befindlichen ISA abläuft. Falls dies nicht der Fall ist, gibt der Kompiler **402** im Schritt **605** einen RTL-Code für den Funktionsaufruf und für das Einlesen des Rückkehrwerts der Funktion in Schritt **613** ab.

Falls der Kompiler **402** in Schritt **602** feststellt, daß die Funktion in einer anderen ISA arbeitet, gibt der Kompiler **402** einen RTL-Code ab, der erforderlich ist, um die Rekonfiguration zu bewirken, und zwar einschließlich des Abspeicherns aller lebenden Register in Schritt **607** und der Durchführung der Rekonfiguration in Schritt **604**. Bei der bevorzugten Ausführungsform handelt es sich bei der RTL-Rekonfigurationsanweisung um keine Standard-RTL-Anweisung, die eine ISA-Identifikation enthält. Der Kompiler **402** gibt dann in Schritt **606** einen RTL-Code für den Funktionsaufruf ab. Der Kompiler **402** gibt dann in Schritt **609** den RTL-Code für die Rekonfiguration zurück an die erste ISA, um in Schritt **611** lebende Register wieder abzuspeichern und um den Rückgabewert der Funktion in Schritt **613** zu lesen.

Bei Beendigung der Schritte **603** oder **613** stellt der Kompiler **402** in Schritt **608** fest, ob eine andere Hochsprachenanweisung berücksichtigt werden muß. Falls dies der Fall ist, kehrt der Kompiler **402** zu Schritt **600** zurück; anderenfalls fährt er mit Schritt **610** fort.

In **Fig.** 3B führt der Kern des Kompilers **402** die Schritte **610** bis **622** aus, um zuvor generierte RTL-Anweisungen in Assemblersprache zu übersetzen.

Der Kompiler **402** wählt dann in Schritt **612** eine nächste RTL-Anweisung innerhalb der augenblicklich berücksichtigten Gruppe von RTL-Anweisungen aus. Der Kompiler **402** erhält in Schritt **618** eine Regel, die eine Weise spezifiziert, in der die augenblickliche Gruppe von RTL-Anweisungen in einen Satz von Maschinenspracheanweisungen übersetzt werden kann, die für die augenblicklich berücksichtigte Gruppe von RTL-Anweisungen existiert. Der Kompiler **402** erzeugt in Schritt **620** einen Satz von Maschinenspracheanweisungen, die entsprechend der Regel der augenblicklich berücksichtigten Gruppe von RTL-Anweisungen entspricht. Der Kompiler **402** stellt dann in Schritt **622** fest, ob eine andere RTL-Anweisung innerhalb des Kontext einer nächsten Gruppe von RTL-Anweisungen berücksichtigt werden muß. Falls dies der Fall ist, kehrt der Kompiler **402** zu Schritt **612** zurück. Anderenfalls führt der Kompiler **402** in Schritt **610** Registerreservierungsschritte (register allocation) aus. Bekanntlich ist eine konsistente Registerarchitektur von einer ISA zur anderen nicht unbedingt erforderlich. Außerdem können gewisse Innenschleifen-ISAs spezielle Register besitzen, für die normale Registerreservierungsvorgänge nicht gelten. Im allgemeinen sind jedoch Außenschleifen-ISAs in der Lage, normale Registerreservierungen zu verwenden.

Beispielsweise könnte der oben angeführte RTL-Code etwa wie folgt in einen Assembler-Code übersetzt werden, wobei der Assembler-Code von ISA zu ISA verschieden wäre:

```
ld y, r3
ld [r3], r0
add 3, r0
St r0, [x].
```

Somit erzeugt der Kompiler **402** wahlweise und automatisch in Entsprechung mit Vielfach-ISAs während Kompilierungsvorgängen Assemblersprachenanweisungen. Oder mit anderen Worten: Während des Kompilierungsvorgangs kompiliert der Kompiler **402** einen Einzelsatz von Programmanweisungen von den Quelldateien **401** entsprechend einer variablen ISA. Bei dem Kompiler **402** handelt es sich vorzugsweise um einen üblichen Kompiler, der modifiziert ist, um die bevorzugten Kompilierungsvorgänge durchzuführen, die zuvor anhand der **Fig.** 3A und 3B beschrieben wurden.

Der Assembler **409** bzw. der Übersetzer für maschinenorientierte Programmiersprache wird betrieben, um Maschinensprachenanweisungen, die vom Kompiler **403** erzeugt wurden, auch dazu zu verwenden, um Objektdateien **403** zu erzeugen. Die Objektdateien **403** werden dann vom Binder bzw. Linker **404** gebunden, der Bitstrom-Speicherstellen und 64-Bit, Bitausgerichtete bzw. Bit-abgeglichene Adressen handhabt, um ein ausführbares Programm **405** zu erzeugen. Das Ladeprogramm **407** verkettet gleiche Segmente von einer Anzahl von Objektdateien **403**, einschließlich von Bitstrom-Segmenten, in ein einzelnes Speicherbild zur Übermittlung an einen rekonfigurierbaren Computer **10**. Bei einer Ausführungsform erfolgt eine solche Verkettung während der Laufzeit bzw. in Echtzeit; bei einer alternativen Ausführungsform erfolgt dies off-line. Es ist vorteilhaft, wenn der Binder **404** in der Lage ist, eine Speicherausrichtung bzw. einen Speicherabgleich auf dem ausführbaren Programm **405** auszuführen, um für die Ausrichtungserfordernisse für den FPGA-

Bitstrom zu sorgen. Gewisse FPGA-Ladeprogrammhardware erfordert Bitströme von konstanter Größe. Deshalb kann der Binder die Speicherausrichtung vornehmen, indem er Bitströme auffüllt, damit die Anforderungen für eine solche Hardware erfüllt werden.

Wenn ein statisches Einbinden (static linking) verwendet wird, werden Bitströme **406** und ausführbare Programme **405** vom Binder **404** zur Zeit des Bindens verbunden. Wenn ein dynamisches Binden verwendet wird, werden die ISA-Bitströme **406** und die ausführbaren Programme **405** zum Zeitpunkt des Ladens des Programms verbunden, so daß das ausführbare Programm **405** und die Bitströme **406** über die Netzwerkverbindung **408** zu dem Ladeprogramm **407** gesendet werden, das auf einen rekonfigurierbaren Computer **10** läuft.

In **Fig.** 3C ist ein Flußdiagramm von weiteren Schritten gezeigt, die zur Erzeugung eines Maschinensprachencodes gemäß einer Ausführungsform der vorliegenden Erfindung ausgeführt werden. Dieses Flußdiagramm gibt im Detail die Zwischendateien an, die erzeugt werden, wenn der RTL-Code in einen maschinenlesbaren Code für einen rekonfigurierbaren Computer übersetzt wird. Der RTL-Code ist mit Bemerkungen versehen **331**, um anzuzeigen, welche ISA sich in Kontext für jede RTL-Anweisung in dem Code befindet. Zu diesem Zeitpunkt werden RTL-Anweisungen modifiziert. Der Kode wird dann in Schritt **332** mit Hilfe von ISA-abhängigen Verfahren und ISA-unabhängigen Verfahren mittels eines Optimierungs-Dienstprogrammes optimiert. Obwohl das Optimierungsdienstprogramm eine ISA-abhängige Optimierung ausführt, verwendet seine Ausgabe maschinenunabhängigen Code. Somit würde die Ausgabe dennoch auf einer beliebigen ISA laufen, obwohl dies nicht notwendiger Weise optimal ist. Schließlich werden maschinenabhängige Befehle im Schritt **333** von dem optimierten Code mittels des Assemblers **409** erzeugt. Dieser Code verwendet abstrakte Register (abstract registers) und andere maschinenabhängige Merkmale. Zusätzliche Schritte zum Aufräumen von Verknüpfungen bzw. Links und weitere unwichtige Schritte können dann ausgeführt werden.

Bei der bevorzugten Ausführungsform der vorliegenden Erfindung umfassen die ISAs eine reconfig-Anweisung, die bewirkt, daß der FPGA von S-Einrichtung **12** einen Bitstrom lädt, auf den mittels eines Parameters der Anweisung verwiesen wird. Somit besitzt jede ISA zumindest einen Programmverschiebungstyp, der mit Programmverschiebungsbitstromadressen in Zusammenhang steht, die als Parameter für die reconfig-Anweisung der ISAs verwendet werden. Die Programmverschiebungseinsprungstelle in der Objektdatei teilt dem Binder mit, die augenblickliche Adresse einer Größe in ein Segment eines ausführbaren Programms zum Zeitpunkt der Bindung zu ersetzen. Programmverschiebungstypen werden nachfolgend ausführlicher beschrieben.

Wie nachfolgend beschrieben wird, werden Bitströme als Datenobjekte definiert, die sich in einem bestimmten Abschnitt befinden, möglicherweise nur zum Lesen, und deshalb sind Standard-Programmverschiebungsverfahren in der Lage, für eine Programmverschiebung von Bitstromadressen zu sorgen, die in analoger Weise mit ISA-reconfig-Anweisungen zu irgendwelchen programmdefinierten, nur lesbaren Daten verwendet werden.

Der rekonfigurierbare Computer **10** führt die Ergebnisse von dem Ladeprogramm aus, das nachfolgend anhand der **Fig.** 9 bis 25B beschrieben wird. Insbesondere erkennt der rekonfigurierbare Computer **10** reconfig-Anweisungen und lädt geeignete ISA-Bitströme, wie sie in Parametern für solche Anweisungen spezifiziert sind.

Erhaltung des Programmzustands

Eine FPGA-Rekonfiguration durch Laden einer neuen ISA kann zu einem Verlust von interner Hardwarezustandsinformation führen. Folglich behält das erfindungsgemäße System und das Verfahren den Programmzustand während einer Rekonfiguration bei, um den Verlust der Ausführungsbefehlsfolge bei solchen Übergängen der Hardware zu vermeiden.

Während der Rekonfiguration verwendet der rekonfigurierbare Computer **10** vorzugsweise einen Aufrufstapel, um irgendwelche Daten abzuspeichern, die erforderlich sein könnten, nachdem die neue ISA geladen worden ist. Ein solcher Speichervorgang wird durch Schieben von Werten auf den Aufrufstapel bewerkstelligt und durch Abspeichern des Stapelzeigers in einer vordefinierten Speicherzelle, die nicht durch die Rekonfiguration beeinflußt werden wird. Im Anschluß an die Rekonfiguration verwendet der rekonfigurierbare Computer **10** den Stapelzeiger, um die zuvor abgespeicherten Werte von dem Aufrufstapel auszuspeichern.

Es sind Stapel in Laufzeitumgebungen für laufende Maschinenprogramme bekannt, die von Hochsprachen kompiliert wurden, die eine Rekursion unterstützen, wie beispielsweise C/C++, Lisp und Pascal. Ein Stapel ist in einem Bereich des Speichers realisiert und der Stapelzeiger (stack pointer; SP) wird in der ISA dazu verwendet, um die Adresse des Beginns des Stapels zu behalten. Ein Wert, beispielsweise die Programmdaten oder die Adresse, wird in dem Stapel gespeichert (oder auf den Stapel "geschoben"), indem der Stapelzeiger dekrementiert wird und der Wert in die in dem Stapelzeiger enthaltene Adresse geschrieben wird. Der Wert wird wieder von dem Stapel abgerufen (oder vom Stapel "heruntergeschoben"), indem der Wert von der in dem Stapelzeigerregister enthaltenen Adresse gelesen wird; dann wird der Stapelzeiger inkrementiert.

Bei der vorliegenden Erfindung wird der dynamische Zustand des Programms, wie beispielsweise lokale Variablen und die Speicherstelle der nächsten Anweisung, die die Hardware ausführen soll, die typischerweise in einem Adreßregister für den nächsten Anweisungszeiger (Next Instruction Pointer Address Register; NIPAR) oder in einen Programmzähler (PC) abgespeichert ist, vor der Rekonfiguration der Hardware auf dem Stapel abgespeichert. Der Stapelzeiger wird bei der vorbestimmten Speicheradresse aufbewahrt. Somit werden die Werte des Stapelzeigers und des NIPARs bei der Hardware-Rekonfiguration aufbewahrt, so daß auf diese später Zugriff genommen werden kann, wenn die Ausführung des Programms beginnt.

In **Fig.** 6 ist ein Flußdiagramm für ein Verfahren zum Konservieren des Programmzustands gemaß der vorliegenden Erfindung gezeigt. Bei Schritt **601** wird eine reconfig-Anweisung empfangen, die anzeigt, daß ein Bitstrom, der eine neue ISA-Konfiguration darstellt, in die Prozessor-Hardware geladen werden soll. Das Argument für die reconfig-Anweisung ist eine physikalische Speicheradresse, die die zu ladende ISA-Konfiguration enthält.

Der Stapelzeiger wird bei Schritt **652** dekrementiert und der NIPAR wird in Schritt **653** in die von dem Stapelzeiger angezeigte Adresse hineingeladen, wodurch NIPAR auf den Stapel geschoben wird. Der Stapelzeiger wird in Schritt **654** unter einer vorbestimmten Adresse im Speicher abgespeichert, die der neuen ISA-Konfiguration bekannt ist. Die neue

ISA-Konfiguration wird dann im Schritt **655** in die Hardware hineingeladen, indem die FPGA **12** dazu veranlaßt wird, den ISA-Bitstrom von einer Speicherstelle in dem Bitstromspeicher **132** zu lesen. Sobald die neue Konfiguration geladen worden ist, lädt diese in Schritt **656** den Stapelzeiger von der bekannten, vorbestimmten Adresse und lädt dann NIPAR von dem Stapel, indem dieser von der Speicherstelle abgerufen wird, die in Schritt **657** in dem Stapelzeiger abgespeichert wurde und dann wird der Stapelzeiger in Schritt **658** inkrementiert. Ein Beispiel für Stapelinhalte während des Rekonfigurationsvorgangs aus **Fig.** 6 wird nachfolgend anhand der **Fig.** 8A bis 8C beschrieben.

Realisierung einer strukturierten Rekonfiguration

Bei einer Ausführungsform der vorliegenden Erfindung wird eine strukturierte Rekonfiguration dadurch bewerkstelligt, daß reconfig-Anweisungen von dem Quellcode in eine Folge von Assemblersprachenanweisungen übersetzt werden. Wie zuvor beschrieben wurde, werden bei der strukturierten Rekonfiguration Rekonfigurations-Übersetzungsanweisungen nur dann zugelassen, wenn eine Funktion aufgerufen wird oder zu einer Funktion zurückgekehrt wird, so daß jede Funktion mit einer einzigen ISA gekennzeichnet ist, die sich während der gesamten Ausführung der Funktion im Kontext befinden soll. In **Fig.** 7 ist ein Flußdiagramm für ein Verfahren zur Realisierung der strukturierten Rekonfiguration gemäß einer Ausführungsform der vorliegenden Erfindung gezeigt.

Das Verfahren gemäß **Fig.** 7 wird ausgeführt, wenn der Aufruf einer Funktion eine aufgerufene Funktion mit sich bringt, die eine ISA-Rekonfigurationsanweisung besitzt. Die S-Einrichtung **12** rettet in Schritt **707** lebende Registerwerte, so daß diese nicht als Folge der Rekonfiguration verlorengehen. Die S-Einrichtung **12** verwendet die augenblickliche ISA, um in Schritt **701** Parameter für die aufgerufene Funktion auf den Stapel zu schieben, weil diese Parameter in Registern vorliegen können. Die S-Einrichtung **12** rekonfiguriert in Schritt **702** in die neue ISA und ruft in Schritt **703** das Unterprogramm auf, das die Funktion darstellt, wobei eine Anweisung, wie beispielsweise "jsr", der ISA der Zielfunktion verwendet wird. Nachdem die aufgerufene Funktion die Ausführung beendet hat, kehrt eine Rückkehranweisung, wie beispielsweise "ret", in Schritt **704** zu der aufrufenden Funktion zurück. S-Einrichtung 12 10 rekonfiguriert in Schritt **705** zu der ursprünglichen ISA für die aufrufende Funktion, liest in Schritt **706** einen Rückgabewert der aufgerufenen Funktion und speichert lebende Registerwerte in Schritt **708** wieder ab. Die Verfahren aus **Fig.** 7 zur Weitergabe von Stapelparametern und zur Rückgabe von Werten können in üblicher Weise realisiert werden, wie sie in nicht rekonfigurierbaren Computern verwendet werden, die keine Parameter oder Rückgabewerte in Register weiterleiten.

Das folgende stellt ein Beispiel für einen Code zur Realisierung einer strukturierten Rekonfiguration dar:

```
#pragma func_isa funcl another_isa
int
funcl(int *i)
{
...
}

#pragma func_isa main isa0
main()
{
int foo, bar;
...
bar = funcl (&foo);
}
```

Zwei Funktionen sind bekannt: main, das ISA0 verwendet, einen Mehrzweck-Anweisungssatz; und funcl, das einen anderen Anweisungssatz (another IS) verwendet, der als another_isa bezeichnet wird. Die #pragma-Anweisungen spezifizieren die Anweisungssätze für die zwei Funktionen.

In einer Ausführungsform der vorliegenden Erfindung, die eine strukturierte Rekonfiguration verwendet, würde der Kompiler **402** die Funktion call bar = funcl (&foo) von dem oben genannten Listing in den folgenden Assemblercode übersetzen. Dabei wurden Kommentare zum Zwecke der Erläuterung hinzugefügt.

```
1 ; schaffe Platz für den Rückgabewert durch Dekrementierung von SP
2 eldi 16, a0
3 esub a0, sp
4 ; berechne die Adresse von Argument foo
5 emov sp, a1
6 eadd a0, a1
7 ; und schiebe diese auf den Stapel
8 estr a1, sp
9 ; reconfig
10 reconfig another_isa
11 ; rufe das Unterprogramm auf
12 jsr funcl
```

```
13 ; gehe zurück zu isa0
14 reconfig isa0
15 ; lösche den geschobenen Parameter
16 eldf d0
17 ; und lese das Ergebnis in die Registervariable bar
18 ldf d0
```

Außerdem könnte Assemblercode zum Abspeichern und wieder Herstellen von lebenden Registerwerten vor der Zeile 1 bzw. nach der Zeile 18 hinzugefügt werden.

In den **Fig.** 8A bis 8C sind Diagramme der Speicherinhalte bei verschiedenen Punkten während der Ausführung des Assemblercodes gezeigt. Die **Fig.** 8A zeigt den Zustand des Stapels **800** nach der Ausführung der Zeilen 1 bis 8 des oben angeführten Assemblercodes. Diese Zeilen bilden den Stapelrahmen, der von func1 verwendet wird. Zunächst wird Platz geschaffen, um einen Rückgabewert abzuspeichern; dann wird die Adresse der Variablen foo auf den Stapel geschoben. Die Speicherstelle **801** enthält die Variable foo im Stapelrahmen für die Funktion main. In diesem Beispiel wird die Variable bar in einem ISA0-Register abgespeichert und erscheint deshalb nicht auf dem Stapel **800**. Die Speicherstelle **802** wird für einen Rückgabewert reserviert und die Speicherstelle **803** enthält die Adresse der Variablen foo.

Fig. 8B zeigt den Zustand des Stapels **800** auf halben Wege während der Ausführung der reconfig-Anweisung bei Zeile 10. Wie man auch durch Vergleich mit **Fig.** 6 erkennen wird, entspricht dieser Zustand des Stapels **800** dem Ende des Schrittes **654**, unmittelbar bevor die neue Konfiguration geladen werden soll. Die augenblickliche Adresse der nächsten Anweisung (NIPAR) wurde auf den Stapel **800** bei der Speicherstelle **804** geschoben und der Stapelzeiger SP wurde auf eine vorbestimmte Adresse (nicht gezeigt) geschrieben. An dieser Stelle werden die Schritte **655** bis **658** ausgeführt, nämlich die Hardware wird rekonfiguriert, der Stapelzeiger wird geladen und NIPAR wird ausgespeichert, wie zuvor beschrieben wurde.

Fig. 8C zeigt den Zustand des Stapels **800** bei der Einsprungstelle zu func1, wobei die jsr func1-Anweisung bei Zeile 12 verwendet wird. Die Speicherstelle **804** enthält nun die Rückkehradresse. Wenn func1 zurückkehrt, rekonfiguriert der Computer **10** zu ISA0 zurück, wird der Parameter &foo von dem Speicher entfernt und wird der Rückkehrwert in die Variable bar gelesen, die der Kompiler **402** für Register d0 reserviert hatte.

Ausführbares Programm und Bindungsformat

Die bevorzugte Ausführungsform der vorliegenden Erfindung erweitert Standardparadigmen zur Softwareentwicklung, um Bitströme mit einzuschließen, die Hardwarekonfigurationen festlegen, die einen mit Hilfe von FPGAs realisierten Computer spezifizieren, der binäre Maschinenanweisungen von einer ausführbaren Datei **405** ausführt. Dies wird dadurch bewerkstelligt, daß ein neues Dateiformat verwendet wird, das als ICARUS ELF bezeichnet wird und das eine Erweiterung des Executable and Linking Format (ELF) umfaßt, das häufig auf Unix-Workstations verwendet wird und in UNIX System Laboratories, Inc., System V Application Binary Interface, 3. Auflage, 1993 beschrieben ist und das hiermit im Wege der Bezugnahme in dieser Patentbeschreibung mit aufgenommen sei.

Wie in dem UNIX System Laboratories, Inc., System V Application Binary Interface, 3. Auflage, 1993 beschrieben ist, handelt es sich bei ELF-Dateien entweder um programmverschiebliche (relocatable) Dateien (Objektdateien **403**) oder um ausführbare Dateien **405**. ELF sorgt für Parallelansichten der Inhalte der Datei, was die differierenden Erfordernisse dieser zwei Formate reflektiert. In **Fig.** 5 ist im Teil **501** ein typisches ELF-Dateiformat in einer Binden-Ansicht und im Teil **502** in einer Ausführen-Ansicht gemäß dem Stand der Technik dargestellt. Der ELF-Kopfteil **503** enthält einen "Plan", der die Organisation der Datei beschreibt. Die Abschnitte **505** beinhalten den Großteil der Information der Objektdatei für die Binden-Betrachtungsweise **501**, einschließlich von Anweisungen, Daten, Symboltabellen, Verschiebungsinformation und dergleichen, wie nachfolgend ausführlicher beschrieben wird. Die Abschnitte **507**, die in der Ausführen-Darstellung **502** verwendet werden, entsprechen den Abschnitten **505**, wobei jeder Abschnitt **507** einem oder mehreren Abschnitten **505** entspricht. Außerdem können die Abschnitte **507** Kopfteile umfassen, die Information enthalten, wie beispielsweise die Information, ob der Abschnitt **507** sich in einem Schreib-Speicher befindet, was auf die Abschnitte **505** anwendbar sein kann oder nicht. Im allgemeinen enthalten die Abschnitte **505** Information, die während des Bindens verwendet wird, während die Abschnitte **507** Information enthalten, die während des Ladens verwendet wird.

Die Programmkopfteiltabelle **504** (falls vorhanden), teilt dem Computer **10** mit, wie ein Verarbeitungsbild aufzubauen ist. Die Abschnittskopfteiltabelle 506 enthält Information, die die Abschnitte **505** beschreibt. Jeder Abschnitt **505** besitzt einen Eintrag in Tabelle 506; jeder Eintrag gibt Information an, wie beispielsweise den Namen des Abschnitts, die Größe und dergleichen. Die in **Fig.** 5 gezeigten Elemente können in einer beliebigen Reihenfolge vorgesehen sein und einige Elemente können fehlen.

Weitere Details, die die in **Fig.** 5 gezeigten Elemente betreffen, kann man in UNIX System Laboratories, Inc., System V Application Binary Interface, 3. Auflage, 1993 finden. Die folgende Beschreibung erklärt die Unterschiede zwischen dem Standard-ELF, wie in System V Application Binary Interface beschrieben, und dem ICARUS ELF-Dateiformat, das bei der vorliegenden Erfindung verwendet wird.

Das ICARUS ELF-Dateiformat verwendet prozessorabhängige Merkmale von ELF, um für eine Verschiebung von Bitstromadressen zu sorgen, die innerhalb des Programmtextes verwendet werden, und um für eine Verschiebung und für ein Binden von Bitströmen in Segmente zu sorgen, die während des Ablaufens des Programms innerhalb eines hierfür vorgesehenen Bitstromspeichers **132** geladen werden können. ICARUS ELF erweitert somit Standard-ELF, um die Abspeicherung von Bitströmen zu erleichtern, die FPGA-Konfigurationen sowie den ausführbaren Code definieren, der auf der FPGA-definierten Hardware läuft.

ICARUS ELF ergänzt den Standard-ELF, um für neue Datentypen, Abschnitte, Symboltypen und Verschiebungstypen für ISA-Bitströme zu sorgen.

Datentypen

Bei der bevorzugten Ausführungsform verwendet der rekonfigurierbare Computer Bitadressen, die 64 Bit breit sind. Die Adressen zeigen auf den Bitversatz des niedrigstwertigen Bits des Daten-Gegenstands. ICARUS ELF ist für 64-Bit Byte-Adressen ausgelegt, wobei die Adresse auf das erste Byte (niedrigstwertig für kleine Endian-Prozessoren, höchstwertig für große Endian-Prozessoren) für jedes Datenelement zeigt. Während die Versätze in Kopfteilen bezüglich der Bytes definiert werden, werden zu verschiebende Adressen in 64-Bit Bit-Adressen spezifiziert. Dies läßt die Verwendung eines Binders auf einem Byte-orientierten Computer zu. ICARUS ELF verwendet zwei neue Datentypen, um eine 64-Bitadressierung zu erleichtern:

- ICARUS_ELF_Addr: Größe-8-Bytes, mit Ausrichtung, die für die augenblickliche ISA durch K_{isa} festgelegt wird, das den Zweier-Logarithmus der Bitbreite des Speichers darstellt (beispielsweise 3 für 8-Bit, 4 für 16-Bit).
- ICARUS_ELF_Off: Byte-Versatz in die Datei, Größe 4 Bytes, Ausrichtung 1 Byte.

Abschnitte

Eine Ausführungsform der vorliegenden Erfindung fügt einen neuen Abschnitt hinzu, der FPGA-Bitstromdaten enthält, mit dem Namen .ICARUS.bitstream. Einer oder mehrere solcher Abschnitte können vorgesehen sein. Bei der bevorzugten Ausführungsform ist jeder solcher Abschnitte vom ELF-Abschnittstyp SHT_PROGBITS und besitzt das ELF-Abschnittsattribut SHF_ALLOC.SHT_PROGBITS bezeichnet einen Abschnitt, der Information enthält, die durch das Programm festgelegt wird, deren Format und Bedeutung ausschließlich über das Programm festgelegt wird. Attribut SHF_ALLOC spezifiziert, daß der Abschnitt während der Vorgangsausführung einen Speicher besetzt, Information, die für das Ladeprogramm nützlich sein kann.

Weil ELF mehrere Beispiele für einen Abschnitt mit einem bestimmten Namen erlaubt, kann die vorliegende Erfindung einen Abschnitt pro Bitstrom verwenden, oder kann alternativ alle Bitströme in einen Abschnitt mit geeigneter Ausrichtung verbinden.

Es ist vorteilhaft, für einen neuen Abschnitt für Bitströme zu sorgen, so daß Hardware mit speziellen Speicherbereichen für Bitströme hergestellt werden kann. Der separate Abschnitt erleichtert die Plazierung von Bitströmen in diesen speziellen Speicherbereichen mit Hilfe des Ladeprogramms. Falls solche Speicherbereiche nicht erforderlich sind, kann die vorliegende Erfindung unter Verwendung eines Standard-Datenabschnitts für Nur-Lese-Programmdaten realisiert werden, wie beispielsweise .rodata und .rodata1, wie in System V Application Binary Interface beschrieben ist, anstatt das spezielle Bitstromabschnitte eingeführt werden.

Symbole

Objektdateien enthalten Symboltabellen, die Information zur Lokalisierung bzw. Fixierung und zur Verschiebung der Symboladressen und Verweise eines Programms halten. In einer Ausführungsform der vorliegenden Erfindung besitzt jeder Bitstrom, der in dem Abschnitt .ICARUS.bitstream enthalten ist, einen Eintrag in der Symboltabelle der Objektdatei. In der Binden-Ansicht **501** aus **Fig.** 5 ist die Symboltabelle in einem separaten Abschnitt **505** lokalisiert. Das Symbol hat die folgenden Attribute:

- st_name: Der Name des Symbols ist der Name, der verwendet wird, um es in der Maschinensprachenquelle für die Objektdatei zu referenzieren. st_name enthält einen Index in die Symbolstringtabelle der Objektdatei, die die Zeichendarstellungen der Symbolnamen enthält.
- st_value: Sorgt bei Bitstrom-Symbolen für den Versatz des Bitstroms innerhalb des Abschnittes.
- st_size: Größe des Bitstroms in Bits.
- st_info: Spezifiziert den Typ und die Binde-Attribute. Ein neuer Typ wird verwendet, der als STT_BITSTREAM bezeichnet wird. Dieser neue Typ ist charakteristisch für die vorliegende Erfindung und zeigt an, daß dieses Symbol sich in einem FPGA-Bitstrom befindet. Der Bindevorgang legt die Sichtbarkeit der Bindung und das Verhalten fest und kann STB_LOCAL oder STB_GLOBAL sein. STB_LOCAL zeigt an, daß das Symbol nicht außerhalb der Objektdatei, die die Definition des Symbols enthält, sichtbar ist. STB-GLOBAL zeigt an, daß das Symbol für alle Dateien, die kombiniert werden, sichtbar ist. Für Bitstrom-Symbole kann das Binden entweder STB_LOCAL oder STB-GLOBAL sein. Weil Bitströme für gewöhnlich von mehr als einem Codeabschnitt verwendet werden und deshalb in eine Bibliothek zur Wiederverwendung übersetzt werden können, ist es wahrscheinlicher, daß STB-GLOBAL verwendet wird.

Verschiebungen (Relocations)

Verschiebung ist der Vorgang der Verbindung symbolischer Referenzen mit symbolischen Definitionen. Verschiebbare Dateien enthalten Verschiebungen bzw. Programmverschiebungen, die Daten darstellen, die beschreiben, wo spezielle symbolische Definitionen gefunden werden können,. so daß der Binder diese lokalisieren kann. Spezielle Verschiebungsvorgänge variieren von ISA zu ISA, wie dies auch bei Standard-ELF-Dateien der Fall ist. Verschiebungstypen sind innerhalb der Felder r_info von ICARUS_ELF_REL-Strukturen und ICARUS_ELF_RELA-Strukturen enthalten. Beispiele für solche Verschiebungstypen umfassen:

- ICARUS_64_BIT_ADDR: 64-Bitadressen, die zum Zeitpunkt der Übersetzung bestimmt werden. Typischer-

weise gemeinsam mit der eldi-Anweisung in den Prozessor geladen.
- ICARUS_64_BIT_OFFSET: Relativadreßversatz von augenblicklicher NIPAR-Stelle zu einem Symbol, typischerweise einem Kennzeichen (label). Von den br-Anweisungen (branch; Programmverzweigung) verwendet.

Aus den genannten Gründen macht erfindungsgemäß das zuvor beschriebene ICARUS ELF-Objektdateiformat neuartigen Gebrauch von der Softwarebindungstechnologie, um Computerprogramme gemeinsam mit der Hardwarekonfiguration zusammenzufügen, auf der das Programm läuft, wobei ein rekonfigurierbarer Computer **10** verwendet wird, wie er zuvor beschrieben wurde. Das System und das Verfahren gemäß der vorliegenden Erfindung kann eine Kompilierung für Mehrfach-ISAs innerhalb einer einzigen Quelldatei ausführen und ist bei einer Ausführungsform in der Lage, Maschinenanweisungen und Daten gemeinsam mit Hardwarekonfigurationen zusammenzufügen, die erforderlich sind, um die Maschineninstruktionen auszuführen.

Nachfolgend wird anhand der **Fig. 9** bis 25B eine bevorzugte Hardware-Umgebung beschrieben, in der die Erfindung bevorzugt angewendet wird.

In **Fig. 9** ist ein Blockdiagramm einer bevorzugten Ausführungsform eines Systems **3010** für ein skalierbares, paralleles, dynamisch rekonfigurierbares Berechnen dargestellt, das gemäß der Erfindung ausgeführt wird. Das in **Fig. 9** gezeigte System **3010** entspricht im wesentlichen dem in **Fig. 1** gezeigten System **10**. Das System **3010** weist vorzugsweise zumindest eine S-Einrichtung **3012**, eine T-Einrichtung **3014** entsprechend jeder S-Einrichtung **3012**, eine universelle Verbindungsmatrix (GPIM) **3016**, zumindest eine Ein-/Ausgabe-T-Einrichtung **3018**, ein oder mehrere Ein-/Ausgabevorrichtungen **3020** und eine Master-Zeitbasiseinheit **3022** auf. In der bevorzugten Ausführungsform weist das System **3010** mehrere S-Einrichtungen **3012** und folglich mehrere T-Einrichtungen **3014** plus mehrere Ein-/Ausgabe-T-Einrichtungen **3018** und mehrere Ein-/Ausgabe-Vorrichtungen **3020** auf.

Jede der S-Einrichtungen **3012**, der T-Einrichtungen **3014** und der Ein-/Ausgabe-T-Einrichtungen **3018** hat einen Master-Zeitsteuereingang, der mit einem Zeitsteuerausgang der Master-Zeitbasiseinheit **3022** verbunden ist. Jede S-Einrichtung **3012** hat einen Eingang und einen Ausgang, der mit der entsprechenden T-Einrichtung **3014** verbunden ist. Zusätzlich zu dem Eingang und dem Ausgang, der mit der entsprechenden S-Einrichtung **3012** verbunden ist, hat jede T-Einrichtung **3014** einen Leiteingang und einen Leitausgang, welche mit der GPI-Matrix **3016** verbunden sind. Dementsprechend hat jede Ein-/Ausgabe-T-Einrichtung **3018** einen Eingang und einen Ausgang, welcher mit einer Ein-/Ausgabe-Vorrichtung **3020** verbunden ist, und einen Leiteingang und einen Leitausgang, der mit der GPI-Matrix **3016** verbunden ist.

Wie unten im einzelnen noch beschrieben wird, ist jede S-Einrichtung **3012** ein dynamisch rekonfigurierbarer Rechner. Die GPI-Matrix **3016** stellt ein paralleles Punkt-zu-Punkt-Verbindungsmittel dar, welches eine Kommunikation zwischen T-Einrichtungen **3014** erleichtert. Der Satz T-Einrichtungen **3014** und die GPI-Matrix **3016** bilden ein paralleles Punkt-zu-Punkt-Verbindungsmittel für einen Datentransfer zwischen S-Einrichtungen **3012**. In ähnlicher Weise bilden die GPI-Matrix **3016**, der Satz T-Einrichtungen **3014** und der Satz Ein-/Ausgabe-T-Einrichtungen **3018** ein paralleles Punkt-zu-Punkt-Verbindungsmittel für einen Ein-/Ausgabe-Transfer zwischen S-Einrichtungen **3012** und jeder Ein-/Ausgabevorrichtung **3020**. Die Master-Zeitbasiseinheit **3022** weist einen Oszillator auf, der ein Master-Zeitsteuersignal zu jeder S-Einrichtung **3012** und jeder T-Einrichtung **3014** schafft.

In einer beispielhaften Ausführungsform ist jede S-Einrichtung **3012** durch Verwenden eines Xilinx C4013 (Xilinx, Inc., San Jose, CA) feldprogrammierbaren Gate-Array (FPGA) ausgeführt, das mit einem 64 Megabyte Randomspeicher (RAM) verbunden ist. Jede T-Einrichtung **3014** ist durch Verwenden von annähernd 50% der rekonfigurierbaren Hardware-Ressourcen in einem Xilinx XC4013 FPGA ausgeführt, ebenso jede Ein-/Ausgabe-T-Einrichtung **3018** ist. Die GPI-Matrix **3014** ist als ein ringförmiges Verbindungsmaschennetz ausgeführt. Die Master-Zeitbasiseinheit **3020** ist ein Taktoszillator, der vorgesehen ist, um eine Verteilungsschaltung zu takten, um eine systemweite Frequenzreferenz zu schaffen. Vorzugsweise übertragen die GPI-Matrix **3014** die T-Einrichtung **3012** und die Ein-/Ausgabe-T-Einrichtung **3018** Information entsprechend ANSI/IEEE-Standard 1596 bis 1992, wodurch ein skalierbares kohärentes Interface (SCI) definiert ist.

In der bevorzugten Ausführungsform weist das System **3010** mehrere S-Einrichtungen **3012** auf, welche parallel arbeiten. Der Aufbau und die Funktionalität jeder der einzelnen S-Einrichtungen **3012** wird im einzelnen anhand von **Fig. 10** bis 20B beschrieben. In **Fig. 10** ist ein Blockdiagramm einer bevorzugten Ausführungsform einer S-Einrichtung **3012** dargestellt. Die S-Einrichtung **3012** weist eine erste lokale Zeitbasiseinheit **3030**, eine dynamisch rekonfigurierbare Verarbeitungs-(DRP-)Einheit **3032** zum Ausführen von Programmbefehlen und einen Speicher **3034** auf. Die erste lokale Zeitbasiseinheit **303** hat einen Zeitsteuereingang, welche den Master-Zeitsteuereingang der S-Einrichtung bildet. Die erste lokale Zeitbasiseinheit **3030** hat auch einen Zeitsteuerausgang, der ein erstes lokales Zeitsteuersignal oder ein Taktsignal an einem Zeitsteuereingang der DRP-Einheit **3032** und an einem Zeitsteuereingang des Speichers **3034** über eine erste Zeitsteuerleitung **3040** schafft. Die DRP-Einheit **3032** hat einen Steuersignal-Ausgang, der mit einem Steuersignaleingang des Speichers **3034** über eine Speichersteuerleitung **3042** verbunden ist, einen Adressenausgang, der mit einem Adresseneingang des Speichers **3034** über eine Adressenleitung **3044** verbunden ist, und einen zweiseitig gerichteten Steuerport, der mit einem zweiseitig gerichteten Steuerport des Speichers **3034** über eine Speicher-Ein-/Ausgabeleitung **3046** verbunden ist. Die DPR-Einheit **3032** hat zusätzlich einen zweiseitig gerichteten Steuerport, der über einen zweiseitig gerichteten Steuerport der entsprechenden T-Einrichtung **3014** über eine externe Steuerleitung **3048** verbunden ist. Wie in **Fig. 10** dargestellt, überspannt die Speichersteuerleitung **3042** X-Bits; die Adressenleitung **3044** überspannt M-Bits; die Speicherein-/Ausgabeleitung **3046** überspannt (N × k) Bits und die externe Steuerleitung **3048** überspannt Y-Bits.

In der bevorzugten Ausführungsform empfängt die erste lokale Zeitbasiseinheit **3030** das Master-Zeitsteuersignal bzw. Master-Taktsignal von der Master-Zeitbasiseinheit **3022**. Die erste lokale Zeitbasiseinheit **3030** erzeugt das erste lokale Zeitsteuersignal auf dem Master-Zeitsteuersignal und gibt das erste lokale Zeitsteuersignal an die DRP-Einheit **3032** und den Speicher **3034** ab. In der bevorzugten Ausführungsform kann sich das erste lokale Zeitsteuersignal von einer S-Einrichtung **3012** zur anderen ändern. Folglich arbeiten die DRP-Einheit **3032** und der Speicher **3034** in einer vorgege-

benen S-Einrichtung **3012** mit einer unabhängigen Taktrate bezüglich der DRP-Einheit **3032** und dem Speicher **3034** in einer anderen S-Einrichtung **3012**. Vorzugsweise ist das erste lokale Zeitsteuersignal phasensynchronisiert mit dem Master-Zeitsteuersignal. In der bevorzugten Ausführungsform ist die erste lokale Zeitbasiseinheit **3030** durch Verwenden einer phasengekoppelten Frequenzumwandlungsschaltung ausgeführt, die eine phasengekoppelte Detektionsschaltung enthält, die mit Hilfe von rekonfigurierbaren Hardware-Ressourcen ausgeführt ist. Der Fachmann weiß, daß in einer alternativen Ausführungsform die erste lokale Zeitbasiseinheit **3030** auch als ein Teil eines Taktverteilungsbaums ausgeführt sein könnte.

Der Speicher **3034** ist vorzugsweise als ein RAM ausgeführt und speichert Programmbefehle, Programmdaten und Konfigurationsdatensätze für die DRP-Einheit **3032**. Der Speicher **3034** einer vorgegebenen S-Einrichtung **3012** ist vorzugsweise für eine andere S-Einrichtung **3012** in dem System **3010** über die GPI-Matrix **3016** zugänglich. Darüber hinaus ist jede S-Einrichtung **3012** vorzugsweise dadurch gekennzeichnet, daß sie einen gleichförmigen Speicheradressenplatz hat. In der bevorzugten Ausführungsform enthalten Programmbefehle, die in dem Speicher **3040** selektiv gespeichert sind, Rekonfigurationsanweisungen, die in Richtung der DRP-Einheit **3032** gerichtet sind.

In **Fig.** 11A weist die beispielhafte Programmauflistung **3050** einen Satz Außenschleifenteile **3052**, sowie erste bis fünfte Innenschleifenteile **3050** bis **3057** auf. Wie der Fachmann weiß, verweist der Begriff "Innenschleife" auf einen iterativen Teil eines Programms, das dafür verantwortlich ist, einen ganz bestimmten Satz verwandter Operationen durchzuführen, und der Begriff "Außenschleife" verweist auf die Teile eines Programms hin, die hauptsächlich dafür verantwortlich sind, universelle Operationen und/oder eine Übertragungssteuerung von einem Innenschleifenteil zu einem anderen durchzuführen. Im allgemeinen führen Innenschleifenteile **3054** bis **3058** eines Programms spezifische Operationen an möglicherweise großen Datensätzen durch. Bei einer Bildverarbeitungsanwendung kann der erste innere Schleifenteil **3054** Farbformat-Umsetzoperationen an Bilddaten durchführen, und die zweiten bis fünften Innenschleifenteile **3055** bis **3058** können eine lineare Filterung, eine Faltung, Mustersuch- und Kompressionsoperationen durchführen. Wie der Fachmann weiß, kann eine aneinanderhängende Folge von Innenschleifenteilen **3055** bis **3058** als eine Software-Pipeline betrachtet werden. Jeder Außenschleifenteil **3052** würde für eine Daten-Ein-/Ausgabe und/oder für ein Leiten der Datenübertragung und ein Steuern von dem ersten Innenschleifenteil **3054** zu dem zweiten Innenschleifenteil **3055** verantwortlich sein. Zusätzlich erkennt der Fachmann, daß ein vorgegebener Innenschleifenteil **3054** bis **3058** eine oder mehrere Rekonfigurationsanweisungen enthalten kann. Im allgemeinen werden für ein vorgegebenes Programm die Außenscheifenteile **3052** der Programmauflistung **3050** eine Vielfalt von universellen Befehlstypen enthalten, während die Innenschleifenteile **3054**, **3056** der Programmauflistung **3050** aus verhältnismäßig wenigen Befehlstypen bestehen, die verwendet werden, um eine spezifische Menge an Operationen durchzuführen.

In einer beispielhaften Programmauflistung **3050** erscheint eine erste Konfigurationsanweisung am Anfang des ersten Innenschleifenteils **3054**, und eine zweite Rekonfigurationsanweisung erscheint am Ende des ersten Innenschleifenteils **3054**. Dementsprechend erscheint eine dritte Konfigurationsanweisung zu Beginn des zweiten Innenschleifenteils **3055**; eine vierte Rekonfigurationsanweisung erscheint zu Beginn des dritten Innenschleifenteils **3056**; eine fünfte Rekonfigurationsanweisung erscheint zu Beginn des vierten Innenschleifenteils **3057** und eine sechste und siebte Rekonfigurationsanweisung erscheint am Anfang bzw. am Ende des fünften Innenschleifenteils **3058**. Jede Rekonfigurationsanweisung verweist vorzugsweise auf einen Rekonfigurationsdatensatz, welcher eine interne DRPU-Hardware-Organisation spezifiziert, die auf die Ausführung einer ganz bestimmen Befehlssatz-Architektur (ISA) gewidmet und dafür optimiert worden ist. Eine IS-Architektur ist ein Stamm- oder Kernsatz von Informationen, die verwendet werden können, um einen Rechner zu programmieren. Eine IS-Architektur definiert Befehlsformate, Operationscodes, Datenformate, Adressiermodes, Ausführungs-Steuerflags und programmzugängliche Register. Der Fachmann weiß, daß dies der herkömmlichen Definition einer IS-Architektur entspricht. In der vorliegenden Erfindung kann jede DRP-Einheit **3032** einer S-Einrichtung schnell laufzeit-konfiguriert werden, um direkt Mehrfach-IS-Architekturen durch die Verwendung eines eindeutigen Konfigurationsdatensatzes für jede gewünschte IS-Architektur auszuführen. Das heißt, jede IS-Achitektur wird mit einer eindeutigen internen DRPU-Hardware-Organisation durchgeführt, wie die durch einen entsprechenden Konfigurationsdatensatz spezifiziert ist. Folglich entsprechen in der vorliegenden Erfindung die ersten bis fünften Innenschleifenteile **3054** bis **3058** jeweils einer eindeutigen IS-Architektur, nämlich ISA 1, 2, 3, 4 bzw. k. Der Fachmann erkennt, daß jede nachfolgende IS-Architektur nicht eindeutig zu sein braucht. Folglich könnte ISA k ISA 1, 2, 3, 4 oder irgendeine andere ISA sein. Der Satz Außenschleifenteile **3052** entspricht auch einer eindeutigen ISA, nämlich ISA 0. In der bevorzugten Ausführungsform kann während einer Programmausführung die Auswahl von aufeinanderfolgenden Rekonfigurationsanweisungen datenabhängig sein. Bei Auswahl einer vorgegebenen Rekonfigurationsanweisung werden Programmbefehle nacheinander gemäß einer entsprechenden IS-Architektur über eine eindeutige DRPU-Hardware-Konfiguration ausgeführt, was durch einen entsprechenden Konfigurations-Datensatz spezifiziert ist.

In der Erfindung kann eine vorgegebene IS-Architektur als eine Innenschleifen-IS-Architektur oder als eine Außenschleifen-IS-Architektur entsprechend der Anzahl und den Typen von Befehlen, welche sie enthält, in Kategorien eingeteilt werden. Eine IS-Architektur, die mehrere Befehle enthält und die zum Durchführen genereller Operationen brauchbar ist, ist eine Außenschleifen-ISA, während eine ISA, die aus relativ wenigen Befehlen besteht und die darauf ausgerichtet ist, spezifische Operationstypen durchzuführen, eine Innenschleifen-ISA ist. Da eine Außenschleifen-ISA darauf gerichtet ist, generelle Operationen durchzuführen, ist eine Außenschleifen-ISA am zweckdienlichsten, wenn eine parallele Programm-Befehlsausführung wünschenswert ist. Die Wirksamkeit einer Ausführung einer Innenschleifen-ISA ist vorzugsweise hinsichtlich Befehlen gekennzeichnet, die pro Taktzyklus durchgeführt werden oder hinsichtlich rechten Ergebnissen gekennzeichnet, die pro Taktzyklus erzeugt worden sind.

Der Fachmann erkennt, daß die vorhergehende Erörterung einer sequentiellen Programmbefehlsausführung und einer parallelen Programmbefehlsausführung eine Programmbefehlsausführung mit einer einzigen DRP-Einheit **3032** betrifft. Das Vorhandensein von mehreren S-Einrichtung Rekonfigurationsanweisungen **3012** in dem System **3010** erleichtert die parallele Ausführung von mehreren Programmbefehlsfolgen in einer vorgegebenen Zeit, wobei jede Programmbefehlsfolge durch eine vorgegebene DRP-Einheit **3032** durchgeführt wird. Jede DRP-Einheit **3032** ist entsprechend konfiguriert, um eine parallele oder serielle Hardware zu haben, um eine ganz bestimmte Innenschleifen-ISA bzw. eine Außen-

schleifen-ISA in einer ganz bestimmten Zeit durchzuführen. Die interne Hardware-Konfiguration einer vorgegebenen DRP-Einheit **3032** ändert sich mit der Zeit entsprechend der Auswahl von einer oder mehreren Rekonfigurationsanweisungen, die in eine Folge von durchzuführenden Programmbefehlen eingebettet sind.

In einer bevorzugten Ausführungsform sind jede IS-Architektur und deren entsprechende interne DRPU-Hardware-Organisation entsprechend ausgelegt, um eine optimale Rechenleistung für eine ganz bestimmte Klasse von Rechenproblemen bezüglich einer Menge verfügbarer rekonfigurierbarer Hardware-Ressourcen zu schaffen. Wie vorher bereits erwähnt und wie nunmehr nachstehend im einzelnen näher beschrieben wird, ist eine interne DRPU-Hardware-Organisation, die einer Außenschleifen-ISA entspricht, vorzugsweise für eine sequentielle Programmbefehlsausführung optimiert, und eine interne DRPU-Hardware-Organisation, die einer Innenschleifen-ISA entspricht, ist vorzugsweise für eine parallele Programmbefehlsausführung optimiert.

Mit Ausnahme jeder Rekonfigurationsanweisung weist die beispielhafte Programmauflistung **3050** der **Fig.** 11A vorzugsweise herkömmliche Hochsprachenangaben auf, beispielsweise Angaben die entsprechend der C-Programmiersprache geschrieben sind. Der Fachmann erkennt, daß das Einbeziehen von einer oder mehreren Rekonfigurationsanweisungen in eine Folge von Programmbefehlen einen Compiler erfordert, der modifiziert ist, um für die Rekonfigurationsanweisungen verantwortlich zu sein.

In **Fig.** 11B ist ein Flußdiagramm von herkömmlichen Compilieroperationen dargestellt, die während des Compilierens bzw. Übersetzens einer Folge von Programmbefehlen durchgeführt worden sind. Hierbei entsprechen die herkömmlichen Compilieroperationen im allgemeinen denjenigen, die von dem GNU C Compiler (GCC) durchgeführt worden sind, der von der Free Software Foundation (Cambridge, MA) hergestellt worden ist. Der Fachmann weiß, daß die herkömmlichen Compilieroperationen, die unten beschrieben werden, ohne weiteres für andere Compiler verallgemeinert werden können. Die herkömmlichen Compilieroperationen beginnen beim Schritt **3500** mit dem Compiler-Frontende, das eine nächste Hochsprachen-Anweisung für eine Folge von Programmbefehlen auswählt. Als nächstes erzeugt das Compiler-Frontende beim Schritt **3502** einen Zwischencode, der der ausgewählten Hochsprachen-Anweisung entspricht, welche im Falle von GCC Register-Transterpegel-(RTL-)Angaben entspricht. Im Anschluß an den Schritt **3502** bestimmt das vordere Compilerende, ob eine andere Hochsprachen-Anweisung eine Beachtung beim Schritt **3504** erfordert. Wenn dem so ist, kehrt das bevorzugte Verfahren auf den Schritt **3500** zurück.

Wenn beim Schritt **3504** das vordere Compilerende bestimmt, daß keine andere Hochsprachenanweisung Beachtung erfordert, führt das hintere Compilerende als nächstes herkömmliche Registerzuordnungsoperationen beim Schritt **3605** durch. Nach dem Schritt **3506** wählt das hintere Compilerende eine nächste RTL-Angabe hinsichtlich einer aktuellen RTL-Anweisungsgruppe beim Schritt **3508** aus. Das hintere Compilerende bestimmt dann, ob eine Vorschrift, die eine Art und Weise spezifiziert, in welcher die aktuelle RTL-Anweisungsgruppe in einen Satz von Assemblersprachen-Anweisungen übersetzt werden kann, beim Schritt **3510** vorhanden ist. Wenn eine derartige Vorschrift nicht vorhanden ist, kehrt das bevorzugte Verfahren auf den Schritt **3508** zurück, um eine andere RTL-Anweisung für ein Einbeziehen in die aktuelle RTL-Anweisungsgruppe auszuwählen. Wenn eine Vorschrift, die der aktuellen RTL-Anweisungsgruppe entspricht, existiert, erzeugt das hintere Compilerende beim Schritt **3512** einen Satz Assemblersprachen-Anweisungen entsprechend der Vorschrift. Nach dem Schritt **3512** stellt das hintere Compilerende fest, ob eine nächste RTL-Anweisung Beachtung im Kontext mit einer nächsten RTL-Anweisungsgruppe erfordert. Wenn dem so ist, kehrt das bevorzugte Verfahren auf Schritt **3508** zurück; andernfalls ist das bevorzugte Verfahren beendet.

Die vorliegende Erfindung enthält vorzugsweise einen Compiler für ein dynamisch rekonfigurierbares Berechnen. In **Fig.** 11C und 11D ist ein Flußdiagramm von bevorzugten Compilieroperationen dargestellt, die von einem Compiler für ein dynamisch rekonfigurierbares Berechnen durchgeführt worden sind. Die bevorzugten Compileroperationen beginnen beim Schritt **3600** mit dem vorderen Ende des Compilers, der eine nächste Hochsprachen-Anweisung in einer Folge von Programmbefehlen auswählt. Als nächstes bestimmt das vordere Ende des Compilers beim Schritt **3602**, ob die ausgewählte Hochsprachen-Anweisung eine Rekonfigurationsanweisung ist. Wenn dem so ist, erzeugt das vordere Ende des Compilers eine RTL-Regkonfigurationsanweisung beim Schritt **3604**, worauf dann das bevorzugte Verfahren auf Schritt **3600** zurückkehrt. In der bevorzugten Ausführungsform ist die RTL-Rekonfigurationsanweisung eine nichtnormierte RTL-Anweisung, die einen ISA-Identifizierung erhält. Wenn beim Schritt **3602** die ausgewählte Hochprogramm-Anweisung nicht eine Rekonfigurationsanweisung ist, erzeugt das vordere Ende des Compilers als nächstes einen Satz RTL-Anweisungen in herkömmlicher Weise beim Schritt **3606**. Nach dem Schritt **3606** bestimmt das vordere Ende des Compilers beim Schritt **3608**, ob eine andere Hochsprachen-Anweisung Berücksichtigung erfordert. Wenn dem so ist, kehrt das bevorzugte Verfahren auf den Schritt **3600** zurück; anderenfalls geht das bevorzugte Verfahren auf Schritt **3610** über, um Operationen am Compilerende zu initiieren.

Beim Schritt **3610** führt das hintere Ende des Compilers für ein dynamisch rekonfigurierbares Berechnen Register-Zuordnungsoperationen durch. In der bevorzugten Ausführungsform der Erfindung ist jede ISA-Architektur so definiert, daß die Register-Architektur von einer IS-Architektur zur anderen folgerichtig ist; daher werden Register-Zuordnungsoperationen in herkömmlicher Weise durchgeführt. Der Fachmann erkennt, daß im allgemeinen eine folgerichtige Register-Architektur von einer ISA zur anderen keine absolute Forderung ist. Als nächstes wählt das hintere Ende des Compilers eine nächste RTL-Anweisung in einer aktuell in Betracht gezogenen RTL-Anweisungsgruppe beim Schritt **3612** aus. Das hintere Ende des Compilers bestimmt dann beim Schritt **3614**, ob die ausgewählte RTL-Anweisung eine RTL-Rekonfigurationsanweisung ist. Wenn die ausgewählte RTL-Anweisung nicht eine RTL-Rekonfigurationsanweisung ist, bestimmt das hintere Ende des Compilers beim Schritt **3618**, ob eine Vorschrift für die aktuell in Betracht gezogene RTL-Anweisungsgruppe existiert. Wenn dem nicht so ist, kehrt das bevorzugte Verfahren auf den Schritt **3612** zurück, um eine nächste RTL-Anweisung für ein Einbeziehen in die aktuell in Betracht gezogene RTL-Anweisungsgruppe zu wählen. In dem Fall, daß eine Vorschrift für die aktuell in Betracht gezogene RTL-Anweisungsgruppe beim Schritt **3618** existiert, erzeugt das hintere Ende des Compilers als nächstes einen Satz Assemblersprachen-Anweisungen beim Schritt **3620**, welche der aktuell in Betracht gezogenen RTL-Anweisungsgruppe gemäß dieser Vorschrift entsprechen. Nach dem Schritt **3620** bestimmt das hintere Ende des Compilers beim Schritt **3622**, ob eine andere RTL-Anweisung eine Berücksichtigung im Kontext mit einer nächsten RTL-Anweisungsgruppe erfordert. Wenn dem so ist, kehrt das bevorzugte Ver-

fahren auf den Schritt **3612** zurück; anderenfalls endet das bevorzugte Verfahren.

Wenn beim Schritt **3614** die ausgewählte RTL-Anweisung eine RTL-Rekonfigurationsanweisung ist, wählt das hintere Ende des Compilers einen Vorschriftensatz beim Schritt **3616** aus, welcher der ISA-Identifizierung der RTL-Rekonfigurationsanweisung entspricht. In der vorliegenden Erfindung existiert vorzugsweise ein eindeutiger Vorschriftensatz für jede ISA. Jeder Vorschriftensatz schafft daher eine oder mehrere Vorschriften, um Gruppen von RTL-Anweisungen in Assemblersprachen-Anweisungen entsprechend einer ganz bestimmten IS-Architektur umzuwandeln. Nach dem Schritt **3616** geht das bevorzugte Verfahren auf Schritt **3618** über. Der Vorschriftensatz, der einer vorgegebenen IS-Architektur entspricht, enthält vorzugsweise ein Vorschrift, um die RTL-Rekonfigurationsanweisung in einen Satz Assemblersprachen-Befehle zu übersetzen, die eine Software-Unterbrechung erzeugen, die auf eine Durchführung eines Rekonfigurations-Abwicklers (handler) hinausläuft, wie im einzelnen unten beschrieben wird.

In der vorstehend beschriebenen Weise erzeugt der Compiler für dynamisch rekonfigurierbares Berechnen selektiv und automatisch Assemblersprachen-Anweisungen entsprechend mehreren IS-Architekturen während Compileroperationen. Mit anderen Worten, während des Compilerprozesses übersetzt der Compiler einen einzigen Satz Programmbefehlen entsprechend einer variablen IS-Architektur. Der Compiler ist vorzugsweise ein herkömmlicher Compiler, der modifiziert worden ist, um bevorzugte Compileroperationen durchzuführen, die vorstehend unter Bezugnahme auf **Fig.** 11C und 11D beschrieben worden sind. Der Fachmann erkennt, daß, obwohl die geforderten Modifikationen nicht komplex sind, solche Modifikationen im Hinblick sowohl auf herkömmliche Compilertechniken als auch im Hinblick auf herkömmliche rekonfigurierbaren Berechnungsmethoden nicht offensichtlich und naheliegend sind.

In **Fig.** 12 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer dynamisch rekonfigurierbaren Verarbeitungseinheit **3032** dargestellt. Die DRP-Einheit **3032** weist eine Befehlsabruf-(IF-)Einheit **3016**, eine Datenoperations-(DO-)Einheit **3062** und eine Adressenoperations-(AO-)Einheit **3064** auf. Jede der IF-Einheiten **3060**, der DO-Einheiten **3062** und der AO-Einheiten **3064** hat einen Zeitsteuereingang, welcher mit der ersten Zeitsteuersignalleitung **3040** verbunden ist. Die IF-Einheit **3060** hat einen Speichersteuerausgang, welcher mit der Leitung **3042** verbunden ist, einen Dateneingang, der mit einer Leitung **3046** verbunden ist, und einen zweigerichteten Steuereingang, der mit der externen Steuerleitung **3048** verbunden ist. Die IF-Einheit **3060** hat zusätzlich einen ersten Steuerausgang, der mit einem ersten Steuereingang der DO-Einheit **3062** über eine erste Steuerleitung **3070** verbunden ist, und einen zweiten Steuerausgang, der mit einem ersten Steuereingang der AO-Einheit **3064** über eine zweite Steuerleitung **3072** verbunden ist. Die IF-Einheit **3060** hat auch einen dritten Steuerausgang, der mit einem zweiten Steuereingang der DO-Einheit **3062** und mit einem zweiten Steuereingang der AO-Einheit **3064** über eine dritte Steuerleitung **3074** verbunden ist. Die DO-Einheit **3062** und die AO-Einheit **3064** haben jeweils einen zweigerichteten Dateneingang, der mit der Speicher-Ein-/Ausgabeleitung **3064** verbunden ist. Schließlich hat die AO-Einheit **3064** einen Adressenausgang, welcher den Adressenausgang der DRP-Einheit bildet.

Die DRP-Einheit **3032** ist vorzugsweise unter Verwendung einer rekonfigurierbaren oder umprogrammierbaren Logikvorrichtung, beispielsweise einer FPGA, wie einer Xilinx XC4013 (Xilinx, Inc., San Jose, CA) oder einer AT&T ORCA™ 1C07 (AT&T Microelectronics, Allentown, PA) durchgeführt. Vorzugsweise schafft die umprogrammierbare Logikvorrichtung eine Anzahl von 1) selektiv umprogrammierbaren Logikblöcken oder konfigurierbaren Logikblöcken CLB), von 2) selektiv programmierbaren Ein-/Ausgabe-Blöcken (IOB), von 3) selektiv umprogrammierbaren Verbindungsstrukturen, von 4) Datenspeicher-Ressourcen, von 5) Puffer-Ressourcen mit drei Zuständen und von 6) verdrahteten logischen Funktionsmerkmalen. Jeder CLB enthält vorzugsweise eine selektiv rekonfigurierbare Schaltungsanordnung zum Erzeugen logischer Funktionen, zum Speichern von Daten und zum Lenken von Signalen. Der Fachmann weiß, daß eine rekonfigurierbare Datenspeicherschaltung auch in einem oder mehreren Datenspeicherblöcken (DSB), die von der Gruppe CL-Blöcken getrennt ist, in Abhängigkeit von der genauen Bemessung der zu verwendenden umprogrammierbaren Logikvorrichtung enthalten sein kann. Hierbei ist angenommen, daß sich die rekonfigurierbare Datenspeicherschaltung in einer FPGA in den CL-Blöcken befindet; das heißt, das Vorhandensein von DS-Blöcken wird nicht angenommen. Der Fachmann erkennt ohne weiteres, daß ein oder mehrere der hier beschriebenen Elemente, die eine auf CL-Blöcken basierende, rekonfigurierbare Datenspeicherschaltung benutzen, eine auf DS-Blöcken basierende Schaltungsanordnung in dem Fall benützen können, daß DS-Blöcke vorhanden sind. Jeder IO-Block enthält vorzugsweise eine selektiv rekonfigurierbare Schaltungsanordnung, um Daten zwischen CL-Blöcken und einem FPAG-Ausgangsanschlußstift zu übertragen. Ein Konfigurationsdatensatz definiert eine DRPU-Hardware-Konfiguration oder Organisation durch Spezifizieren von Funktionen, die in W-Blöcken durchgeführt worden sind, sowie Verbindungen 1) in CL-Blökken, 2) zwischen CL-Blöcken, 3) in IO-Blöcken, 4) zwischen IO-Blöcken und 5) zwischen CL- und IO-Blöcken. Der Fachmann erkennt, daß über einen Konfigurationsdatensatz, die Anzahl Bits jeweils in der Speichersteuerleitung **3042**, der Adressenleitung **3044**, der Speicherleitung **3046** und der externen Steuerleitung **3048** rekonfigurierbar ist. Vorzugsweise werden Konfigurationsdatensätze in einem oder mehreren Speichern **3034** der S-Einrichtung in dem System **3010** gespeichert. Der Fachmann erkennt, daß die DRP-Einheit **3032** nicht auf eine auf FPGA basierende Ausführung beschränkt ist. Beispielsweise könnte die DRP-Einheit **3032** als eine auf einem RAM basierende Zustandseinrichtung ausgeführt sein, die möglicherweise ein oder mehrere Hinweistabellen enthält. Alternativ hierzu könnte die DRP-Einheit **3032** mit Hilfe einer komplexen programmierbaren Logikvorrichtung (CPLD) ausgeführt sein. Jedoch wird der Fachmann realisieren, daß einige der S-Einrichtungen **3012** des Systems **3010** DRP-Einheiten **3032** haben können, die nicht rekonfigurierbar sind.

In einer bevorzugten Ausführungsform sind die IF-Einheit **3060**, die DO-Einheit **3062** und die AO-Einheit **3064** jeweils dynamisch rekonfigurierbar. Folglich kann deren interne Hardware-Konfiguration selektiv während einer Programmdurchführung modifiziert werden. Die IF-Einheit **3060** richtet Befehlsabruf- und Decodieroperationen, Zugriffsoperationen, DRPU-Rekonfigurationsoperationen aus und gibt Steuersignale an die DO-Einheit **3062** und die AO-Einheit **3064** ab, um eine Befehlsausführung zu erleichtern. Die DO-Einheit **3062** führt Operationen einschließlich einer Datenberechnung durch und die AO-Einheit **3064** führt Operationen einschließlich einer Adressenberechnung durch. Der innere Aufbau und die Arbeitsweise der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** wird nunmehr im einzelnen beschrieben.

In **Fig.** 13 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer Befehlsabruf-(IF-)Einheit **3060** dargestellt. Die IF-Einheit **3060** weist einen Befehlszustands-Zuordner(ISS) **3100**, einen Architekturbeschreibungsspeicher **3101**, eine Speicherzugriffslogik **3102**, einen Rekonfigurationslogik **3104**, eine Unterbrechungslogik **3106**, eine Abrufsteuereinheit **3108**, einen Befehlspuffer **3110**, eine Decodiersteuereinheit **3112**, einen Befehlsdecodierer **3114**, einen Operationscode-Speicherregistersatz **3116**, einen Registerdatei-(RF-)Adressenregistersatz **3118**, einen Konstanten-Registersatz **3120** und einen Prozeßsteuer-Registersatz **3122** auf. Der IS-Zuordner **3100** hat einen ersten und zweiten Steuerausgang, welcher die ersten und zweiten Steuerausgänge der IF-Einheit bildet und einen Zeitsteuereingang, welcher den Zeitsteuereingang der IF-Einheit bildet. Der IS-Zuordner **3100** hat auch einen Abruf/Decodier-Steuerausgang, der mit einem Steuereingang der Abrufsteuereinheit **3108** und einem Steuereingang der Decodiersteuereinheit **3112** über eine Abruf-Decodier-Steuerleitung **3130** verbunden ist. Der IS-Zuordner **3100** hat zusätzlich einen zweigerichteten Steuereingang, der mit einem ersten zweigerichteten Steuereingang jeweils der Speicherzugrifflogik **3102**, der Rekonfigurationslogik **3104** und der Unterbrechungslogik **3106** über eine zweigerichtete Steuerleitung **3132** verbunden ist. Der IS-Zuordner **3100** hat auch einen Operationscode-Eingang, der mit einem Ausgang des Operationscode-Speicherregistersatzes **3116** über eine Operationscode-Leitung **3142** verbunden ist. Schließlich hat der IS-Zuordner **3100** einen zweigerichteten Dateneingang, der mit einem zweigerichteten Dateneingang des Prozeßsteuer-Registersatzes **3122** über eine Prozeßdatenleitung **3144** verbunden ist.

Jeweils die Speicherzugriffslogik **3102**, die Rekonfigurationslogik **3104** und die Unterbrechungslogik **3106** haben einen zweiten zweigerichteten Steuereingang, welcher mit der externen Steuerleitung **3048** verbunden ist. Die Speicherzugriffslogik **3102**, die Rekonfigurationslogik **3104** und die Unterbrechungslogik **3106** haben zusätzlich jeweils einen Dateneingang, der mit einem Datenausgang des Architektur-Beschreibungsspeichers **3101** über eine Steuerleitung **3138** verbunden ist. Die Speicherzugriffslogik **3102** hat zusätzlich einen Steuerausgang, welcher den Speichersteuerausgang der IF-Einheit bildet, und die Unterbrechungslogik **3106** hat zusätzlich einen Ausgang, der mit der Prozeßdatenleitung **3144** verbunden ist. Der Befehlspuffer **3110** hat einen Dateneingang, welcher den Dateneingang der IF-Einheit bildet, einen Steuereingang, der mit einem Steuerausgang der Abrufsteuereinheit **3108** über eine Abrufsteuerleitung **3134** verbunden ist, und einen Ausgang, der mit einem Eingang des Befehlsdecodierers **3114** über eine Befehlsleitung **3136** verbunden ist. Der Befehlscodierer **3114** hat einen Steuereingang, der mit einem Steuerausgang der Decodiersteuereinheit **3112** über eine Decodiersteuerleitung **3138** verbunden ist, und einen Ausgang, der über eine Befehlsleitung **3140** mit 1) einem Ausgang des Operationscode-Speicherregisters **3116**, mit 2) einem Eingang des HF-Adressenregistersatzes **3118** und 3) mit einem Eingang des Konstanten-Registersatzes **3120** verbunden ist. Der HF-Adressenregistersatz **3118** und der Konstanten-Registersatz **3120** haben jeweils einen Ausgang, die zusammen den dritten Steuerausgang **3074** der IF-Einheit bilden.

Der Architektur-Beschreibungsspeicher **3101** speichert Architektur-Spezifikationssignale, welche die aktuelle DRPU-Konfiguration kennzeichnen. Vorzugsweise enthalten die Architektur-Spezifikationssignale 1) eine Referenz zu einem fehlerhaften Konfigurationsdatensatz, 2) eine Referenz zu einer Liste zulässiger Konfigurationsdatensätze, 3) eine Referenz zu einem Konfigurationsdatensatz, welcher der aktuell in Betracht gezogenen ISA entspricht, d. h. eine Referenz zu dem Konfiguration-Datensatz, welcher die aktuelle DRPU-Konfiguration definiert, 4) eine Verbindungs-Adressenliste, welche eine oder mehrere Verbindungs-Ein/Ausgabeeinheiten **3304** in der T-Einrichtung **3014** identifiziert, welche der S-Einrichtung **3012** zugeordnet ist, in welcher die IF-Einheit **3060** untergebracht ist, wie im einzelnen unten in Verbindung mit **Fig.** 21 noch beschrieben wird, 5) einen Satz Unterbrechungsansprechsignale, welche eine Unterbrechungslatenz und Unterbrechungspräzisions-Information spezifizieren, welche definiert, wie die IF-Einheit **3060** auf Unterbrechungen antwortet, und 6) eine Speicherzugriffskonstante, welche ein Speicheradressen-Inkrement definiert.

In der bevorzugten Ausführungsform implementiert jeder Konfigurations-Datensatz den Architektur-Beschreibungsspeicher **3101** als einen Satz von CL-Blöcken, die als ein Festwertspeicher (ROM) konfiguriert sind. Die Architektur-Spezifikationssignale, welche den Inhalt des Architekturbeschreibungsspeichers **3101** definieren, sind vorzugsweise in jedem Konfigurations-Datensatz enthalten. Folglich ändert sich, da jeder Konfigurations-Datensatz einer ganz bestimmten ISA entspricht, der Inhalt des Architektur-Beschreibungsspeichers **3103** entsprechend der aktuellen in Betracht gezogenen ISA. Für eine vorgegebene ISA wird ein Programmzugriff auf den Inhalt des Architektur-Beschreibungsspeichers **3101** vorzugsweise durch das Einbeziehen eines Speicherlesebefehls in dem ISA erleichtert. Dies ermöglicht einem Programm, die Information über die aktuelle DRPU-Konfiguration während einer Programmdurchführung wieder aufzufinden.

In der vorliegenden Erfindung ist die Konfigurationslogik **3104** eine Zustandseinrichtung, welche eine Folge von Rekonfigurationsoperationen steuert, welche eine Rekonfiguration der DRP-Einheit **3032** entsprechend einem Konfigurationsdatensatz erleichtern. Vorzugsweise initiiert die Rekonfigurationslogik **3104** die Rekonfigurationsoperationen bei Empfang eines Rekonfigurationssignals. Wie unten noch im einzelnen beschrieben wird, wird das Rekonfigurationssignal von der Unterbrechungslogik **3106** entsprechend einer Rekonfigurationsunterbrechung, die auf der externen Steuerleitung **3048** erhalten worden ist, oder durch die ISS **3100** entsprechend einer Rekonfigurationsanweisung erzeugt, die in ein Programm eingebettet ist. Die Rekonfigurationsoperationen sorgen für eine anfängliche DRPU-Konfiguration, auf die eine Energieeinschalt/Rücksetzbedingung mit Hilfe des vorgegebenen Konfigurationsdatensatzes von dem Architektur-Beschreibungsspeicher **3101** folgt. Die Rekonfigurationsoperationen sorgen auch für eine selektive DRPU-Rekonfiguration, nachdem die anfängliche DRPU-Konfiguration eingerichtet worden ist. Bei Beendigung der Rekonfigurationsoperationen gibt die Rekonfigurationslogik **3104** ein Beendigungssignal ab. In der bevorzugten Ausführungsform ist die Rekonfigurationslogik **3104** eine nichtrekonfigurierbare Logik, welche das Laden von Konfigurationsdatensätze in die umprogrammierbare Logikvorrichtung selbst steuert, und folglich ist die Folge von Rekonfigurationsoperationen von dem Hersteller der umprogrammierbaren Logikvorrichtung festgelegt. Die Rekonfigurationsoperationen sind daher dem Fachmann bekannt.

Jede DRPU-Konfiguration ist vorzugsweise durch einen Konfigurations-Datensatz gegeben, welcher eine ganz bestimmte Hardware-Organisation festlegt, die der Ausführung einer entsprechenden ISA zugeeignet ist. In der bevorzugten Ausführungsform enthält die IF-Einheit **3060** jeweils die vorstehend angegebenen Elemente unabhängig von der

DRPU-Konfiguration. Bei einem Grundpegel ist die Funktionalität, die durch jedes Element in der IF-Einheit **3060** geschaffen ist, unabhängig von der aktuell in Betracht gezogenen ISA. Jedoch können sich in der bevorzugten Ausführungsform der detaillierte Aufbau und die Funktionalität eines oder mehrerer Elemente der IF-Einheit **3060** basierend auf der Beschaffenheit der ISA ändern, für welche sie konfiguriert worden ist. In der bevorzugten Ausführungsform bleiben der Aufbau und die Funktionalität des Architektur-Beschreibungsspeichers **3101** und der Rekonfigurationslogik **3104** vorzugsweise von einer DRPU-Konfiguration zur anderen konstant. Der Aufbau und die Funktionalität der anderen Elemente der IF-Einheit **3060** und der Art und Weise, in welcher sie sich entsprechend dem ISA-Typ ändern, wird nunmehr im einzelnen beschrieben.

Der Prozeßsteuerregister-(PCR-)Satz **3122** speichert Signale und Daten, die von dem ISS **3100** während einer Befehlsausführung verwendet worden sind. In der bevorzugten Ausführungsform weist der PCR-Satz **3122** ein Register zum Speichern eines Prozeßsteuerwarts, ein Register zum Speichern eines Unterbrechungsvektors und ein Register zum Speicher einer Referenz zu einem Konfigurationsdatensatz auf. Das Prozeßsteuerwort enthält vorzugsweise eine Anzahl von Bedingungsflags, die selektiv basierend auf Bedingungen gesetzt und rückgesetzt werden können, die während einer Befehlsausführung auftreten. Das Prozeßsteuerwort enthält zusätzlich eine Anzahl von Übergangssteuersignalen, die ein oder mehrere Arten festlegen, auf welchen Unterbrechungen verwaltet werden können, wie im einzelnen unten noch beschrieben wird. In der bevorzugten Ausführungsform ist der PCR-Satz **3122** als ein Satz von LC-Blöcken ausgeführt, die für eine Datenspeicher- und Ansteuerlogik konfiguriert sind.

ISS **3100** ist vorzugsweise eine Zustandseinrichtung, welche die Operation der Abrufsteuereinheit **3108**, der Decodiersteuereinheit **3112**, der DO-Einheit **3062** und der AO-Einheit **3064** steuert und Speicherlese- und -Schreibsignale an die Speicheradressenlogik **3102** abgibt, um eine Befehlsausführung zu erleichtern. In **Fig.** 14 ist ein Zustandsdiagramm dargestellt, das eine bevorzugte Menge von Zuständen zeigt, die von der ISS **3100** gestützt sind. Im Anschluß an eine Energieeinschalt- oder Rücksetzbedingung oder unmittelbar nachdem eine Rekonfiguration aufgetreten ist, beginnt die ISS **3100** eine Operation im Zustand P. Entsprechend dem Beendigungssignal, das von der Rekonfigurationslogik **3104** abgegeben worden ist, geht die ISST- **3100** auf den Zustand S über, in welchem der ISS Programmzustandsinformation für den Fall initialisiert oder wieder speichert, daß eine Energieeinschalt-/Rücksetzbedingung oder eine Rekonfiguration aufgetreten ist.

Die ISS **3100** rückt als nächstes auf einen Zustand F vor, in welchem Befehlsabruf-Operationen durchgeführt werden. Bei den Befehlsabrufoperationen gibt ISS **3100** ein Speicherlesesignal an die Speicherzugrifflogik **3102** ab, gibt ein Abrufsignal an die Abrufsteuereinheit **3108** und ein Inkrementsignal an die AO-Einheit **3064** ab, um ein nächstes Befehlsprogrannadressen-(NIPA-)Register **3232** zu inkrementieren, wie im einzelnen unter Bezugnahme auf **Fig.** 19A und 19B nachstehend noch beschrieben wird. Nach dem Zustand F geht ISS **3100** auf einen Zustand D vor, um Befehlsdecodieroperationen zu initiieren. Im Zustand D gibt ISS **3100** ein Decodiersignal an die Decodiersteuerlogik **3112** ab, während beim Zustand D ISS **3100** zusätzlich einen Operationscode wiederfindet, der einem decodierten Befehl von dem Operationscode-Speicherregistersatz **3116** entspricht.

Basierend auf dem wiedergefundenen von dem Operationscode geht ISS **3100** auf einen Zustand E oder M über, um Befehlsausführungsoperationen durchzuführen. ISS **3100** geht auf den Zustand E in dem Fall vor, daß der Befehl in einem einzigen Taktzyklus ausgeführt werden kann; anderenfalls geht ISS **3100** auf den Zustand M für eine in Mehrzyklen durchzuführende Befehlsausführung vor. Bei den Befehlsausführungsoperationen erzeugt ISS **3100** DOU und AOU-Steuersignale und/oder Signale, die auf die Speicherzugriffslogik **3102** gerichtet sind, um die Ausführung des Befehls zu erleichtern, der dem wiederaufgefundenen Operationscode entspricht. Im Anschluß entweder an den Zustand E oder M geht ISS **3100** auf einen Zustand W über.

Im Zustand W erzeugt ISS **3100** DOU-, AOU-Steuersignale und/oder Speicherschreibsignale, um ein Speichern eines Befehlsausführungsergebnisses zu erleichtern. Der Zustand W wird daher als ein Rückschreibzustand bezeichnet. Der Fachmann weiß, daß die Zustände F, D, E oder M und W einen vollständigen Befehlsausführungszyklus aufweisen. Nach dem Zustand W geht die ISS **3100** auf den Zustand Y in dem Fall über, daß eine Suspension einer Befehlsausführung gefordert wird. Der Zustand Y entspricht einem Freizustand, welcher gefordert werden kann, beispielsweise in dem Fall, daß eine T-Einrichtung **3014** Zugriff auf den Speicher **3034** der S-Einrichtung fordert. Nach dem Zustand Y oder W geht in dem Fall, daß eine Befehlsausführung fortzusetzen ist, die ISS **3100** auf den Zustand F zurück, um wieder einen anderen Befehlsausführungszyklus aufzunehmen.

Wie in **Fig.** 14 dargestellt, weist das Zustandsdiagramm auch einen Zustand I auf, welcher als ein Unterbrechungsservice-Zustand definiert ist. Bei der Erfindung empfängt ISS **3100** Unterbrechungsankündigungssignale von der Unterbrechungslogik **3106**. Wie im einzelnen unter Bezugnahme auf **Fig.** 15 auch beschrieben wird, erzeugt die Unterbrechungslogik **3106** Übergangssteuersignale und speichert diese Signale in dem Prozeßsteuerwort in dem PCR-Satz **3122**. Die Übergangssteuersignale zeigen vorzugsweise an, welcher der Zustände F, D, E, M, W und Y unterbrechbar sind und zeigen einen Pegel einer Unterbrechungspräzision, die in jedem unterbrechbaren Zustand erforderlich ist, und für jeden unterbrechbaren Zustand einen nächsten Zustand an, bei welchem eine Befehlsdurchführung im Anschluß an den Zustand I fortzusetzen ist. Wenn ISS **3100** ein Unterbrechungsankündigungssignal erhält, während es sich in einem vorgegebenen Zustand befindet, geht ISS **3100** auf den Zustand I über, wenn die Übergangssteuersignale anzeigen, daß der aktuelle Zustand unterbrechbar ist. Andernfalls geht ISS **3100** weiter, wenn kein Unterbrechungssignal empfangen worden ist, bis ein unterbrechbarer Zustand erreicht wird.

Sobald ISS **3100** auf den Zustand I vorgerückt ist, greift ISS **3100** vorzugsweise auf den PCR-Satz **3122** zu, um ein Unterbrechungsabdeckflag zu setzen und einen Unterbrechungsvektor wieder aufzufinden. Nach dem Wiederauffinden des Unterbrechungsvektors verwaltet ISS **3100** vorzugsweise die aktuelle Unterbrechung über einen herkömmlichen Subroutine-Sprung zu einem Unterbrechungshandhabungsprogramm, wie es durch den Unterbrechungsvektor spezifiziert ist.

In der Erfindung wird eine Rekonfiguration der DRPU-Einheit **3032** entsprechend 1) einer Rekonfigurationsunterbrechung, die auf der externen Steuerleitung **3048** durchgesetzt oder 2) die Durchführung einer Rekonfigurationsanweisung in einer Folge von Programmbefehlen initiiert. In der bevorzugten Ausführungsform führen sowohl die Konfigurations-

unterbrechung als auch die Durchführung einer Rekonfigurationsanweisung zu einem Subroutine-Sprung auf ein Rekonfigurations-Handhabungsprogramm. Vorzugsweise stellt das Rekonfigurations-Handhabungsprogramm eine Programmzustandsinformation sicher und gibt eine Konfigurationsdatensatz-Adresse und das Rekonfigurationssignal an die Rekonfigurationslogik **3104** ab.

Für den Fall, daß die aktuelle Unterbrechung nicht eine Rekonfigurationsunterbrechung ist, geht ISS **3100** auf einen nächsten Zustand über, was durch die Übergangssteuersignale angezeigt ist, sobald die Unterbrechung verwaltet worden ist, um dadurch einen Befehlsdurchführungszyklus wieder aufzunehmen, Befehle durchzuführen oder zu initiieren.

In der bevorzugten Ausführungsform ändert sich die Menge an Zuständen, welche durch ISS **3100** gestützt sind, entsprechend der Beschaffenheit der ISA, für welche die DRP-Einheit **3032** konfiguriert worden ist. Folglich ist der Zustand M für eine ISA nicht vorhanden, in welcher ein oder mehrere Befehle in einem einzigen Taktzyklus durchgeführt werden können, wie es der Fall bei einer typischen Innenschleifen-IS-Architektur ist. Wie beschrieben, definiert das Zustandsdiagramm der **Fig.** 14 vorzugsweise die Zustände, die von ISS **3100** gestützt worden sind, um eine generelle Außenschleifen-IS-Architektur durchzuführen. Für das Durchführen einer Innenschleifen-ISA trägt ISS **3100** vorzugsweise mehrere Sätze von Zuständen F, D, E und W parallel nebeneinander, um dadurch eine Pipeline-Steuerung einer Befehlsdurchführung in einer Weise zu erleichtern, die dem Fachmann ohne weiteres verständlich ist. In der bevorzugten Ausführungsform ist ISS **3100** als eine auf einem CL-Block basierende Zustandseinrichtung ausgeführt, welche die Zustände oder eine Untergruppe der vorstehend beschriebenen Zustände entsprechend einem aktuell in Betracht gezogenen ISA unterstützt.

Die Unterbrechungslogik **3106** weist vorzugsweise eine Zustandseinrichtung auf, welche Übergangssteuersignale erzeugt und Unterbrechungshinweis-Operationen entsprechend einem Unterbrechungssignal durchführt, das über die externe Steuerleitung **3048** erhalten worden ist. In **Fig.** 15 ist ein Zustandsdiagramm dargestellt, das einen bevorzugten Satz Zustände zeigt, die von der Unterbrechungslogik **3106** gestützt sind. Die Unterbrechungslogik **3106** beginnt eine Operation im Zustand P. Der Zustand P entspricht einer Energieeinschalt-Rücksetz- oder Rekonfigurationsbedingung. Entsprechend dem Beendigungssignal, das von der Rekonfigurationslogik **3104** abgegeben worden ist, geht die Unterbrechungslogik **3106** auf den Zustand A über und findet die Unterbrechungsantwortsignale aus dem Architekturbeschreibungsspeicher **3101** wieder. Die Unterbrechungslogik **3106** erzeugt dann die Übergangssteuersignale aus den Unterbrechungsantwortsignalen und speichert die Übergangssteuersignale in dem Prozeßsteuerregister-(PCR-)Satz **3122**. In der bevorzugten Ausführungsform enthält die Unterbrechungslogik **3106** eine auf CLB basierende programmierbare Logikanordnung (PLA), um die Unterbrechungsantwortsignale zu empfangen und um die Übergangssteuersignale zu erzeugen. Im Anschluß an den Zustand A geht die Unterbrechungslogik **3106** auf den Zustand B über, um auf ein Unterbrechungssignal zu warten. Bei Empfang eines Unterbrechungssignals geht die Unterbrechungslogik **3106** auf einen Zustand C in dem Fall über, daß das Unterbrechungsabdeckflag in dem PCR-Satz **3122** rückgesetzt wird. Sobald sie im Zustand C ist, bestimmt die Unterbrechungslogik **3106** den Ursprung der Unterbrechung, eine Unterbrechungspriorität und die Unterbrechungs-Handhabungsadresse. In dem Fall, daß das Unterbrechungssignal eine Rekonfigurationsunterbrechung ist, geht die Unterbrechungslogik **3106** auf einen Zustand R über, und speichert eine Konfigurationsdatensatz-Adresse in dem PCR-Satz **3122**. Nach dem Zustand R oder im Anschluß an den Zustand G für den Fall, daß das Unterbrechungssignal keine Rekonfigurations-Unterbrechung ist, geht die Unterbrechungslogik **3106** auf einen Zustand N über und speichert die Unterbrechungshandhabungsadresse in dem PCR-Satz **3122**. Die Logik **3106** geht auf einen Zustand X über und gibt ein Unterbrechungshinweissignal an ISS **3100** ab. Im Anschluß an den Zustand X kehrt die Unterbrechungslogik **3122** auf den Zustand B zurück, um auf ein nächstes Unterbrechungssignal zu warten.

In der bevorzugten Ausführungsform ändert sich der Pegel einer Unterbrechungslatenz, wie sie durch die Unterbrechungsantwortsignale und folglich durch die Übergangssteuersignale spezifiziert ist, entsprechend einer aktuellen ISA, für welche die DRP-Einheit **3032** konfiguriert worden ist. Beispielsweise erfordert eine ISA, die einer Hochleistungs-Echtzeitbewegungssteuerung gewidmet ist, schnelle und vorhersehbare Unterbrechungsantwort-Fähigkeiten. Der Konfigurationsdatensatz, der einer solchen ISA entspricht, weist daher vorzugsweise Unterbrechungsanwortsignale auf, die anzeigen, daß eine Unterbrechung geringer Latenz erforderlich ist. Die entsprechenden Übergangssteuersignale wiederum identifizieren vorzugsweise mehrfache ISS-Zustände als unterbrechbar, um dadurch eine Unterbrechung zuzulassen, um einen Befehlsausführungszyklus vor der Beendigung der Befehlsausführungszyklus aufzuschieben. Im Gegensatz zu einer ISA für eine Realzeit-Bewegungssteuerung erfordert eine ISA für Bildfaltungsoperationen Unterbrechungsantwort Möglichkeiten, die sicherstellen, daß die Anzahl Faltungsoperationen, die pro Zeiteinheit durchgeführt worden sind, maximiert ist. Der Konfigurationsdatensatz, welcher der Bildfaltungs-ISA entspricht, weist vorzugsweise Unterbrechungsantwortsignale auf, die eine hochlatente Unterbrechung erforderlichenfalls spezifizieren. Die entsprechenden Übergangssteuersignale identifizieren vorzugsweise einen Zustand W, wenn er unterbrechbar ist. Für den Fall, daß ISS **3100** Mehrfachsätze von Zuständen F, D, E und W in paralleler Form stützt, wenn konfiguriert wird, um die Bildfaltungs-ISA durchzuführen, identifizieren die Übergangssteuersignale vorzugsweise jeden Zustand W, wenn er unterbrechbar ist, und spezifizieren ferner, daß eine Unterbrechungsbedienung zu verzögern ist, bis jeder der parallelen Befehlsdurchführungszyklen, deren den Zustand W betreffenden Operationen beendet haben. Hierdurch ist sichergestellt, daß eine ganze Gruppe von Befehlen durch geführt wird, bevor eine Unterbrechung vorgenommen wird, um dadurch vernünftige Pipeline-Durchführungs-Leistungspegel aufrechtzuerhalten.

Analog zu dem Unterbrechungslatenz-Pegel ändert sich auch der Pegel einer Unterbrechungspräzision, wie sie durch die Unterbrechungsantwortsignale spezifiziert ist, entsprechend der ISA, für welche die DRP-Einheit **3032** konfiguriert wird. Beispielsweise spezifizieren in dem Fall, daß der Zustand M als ein unterbrechbarer Zustand für eine Außenschleifen-ISA definiert ist, welche unterbrechbare Mehrzyklen-Operationen stützt, vorzugsweise die Unterbrechungsantwortsignale, daß genau Unterbrechungen gefordert werden. Die Übergangssteuersignale spezifizieren dann, daß Unterbrechungen, die beim Zustand M erhalten worden sind, als genaue Unterbrechungen behandelt werden, um sicherzustellen, daß Mehrzyklen-Operationen erfolglich wieder gestartet werden können. Bei einem anderen Beispiel spezifizieren für eine ISA, welche nicht-fehlerhafte, durch eine Pipeline verbundene Rechenoperationen stützt, die Unterbrechungsantwortsignale vorzugsweise, daß nicht genaue Unterbrechungen gefordert werden. Die Übergangssteuersignale spezifizie-

ren dann, daß Unterbrechungen, die beim Zustand W erhalten worden sind, als ungenaue Unterbrechungen behandelt werden.

Für eine vorgegebene ISA werden die Unterbrechungsantwortsignale definiert oder durch einen Teil des entsprechenden Konfigurationsdatensatzes einer ISA programmiert. Über die programmierbaren Unterbrechungsantwortsignale und die Erzeugung von entsprechenden Übergangssteuersignalen erleichtert die Erfindung, die Durchführung eines optimalen Unterbrechungsschemas auf einer ISA-zu-ISA-Basis. Der Fachmann weiß, daß die große Mehrzahl von herkömmlichen Computer-Architekturen nicht für die flexible Spezifikation von Unterbrechungsmöglichkeiten, nämlich einer programmierbaren Zustandsübergangsfreigabe, einer programmierbaren Unterbrechungslatenz und einer programmierbaren Unterbrechungspräzision vorsorgen. In der bevorzugten Ausführungsform ist die Unterbrechungslogik **3106** als eine auf CLB basierende Zustandseinrichtung ausgeführt, welche die vorstehend beschriebenen Zustände stützt.

Die Abrufsteuereinheit **3108** richtet das Laden von Befehlen in den Befehlspuffer **3110** entsprechend dem Abrufsignal aus, das von ISS **3100** abgegeben worden ist und in der bevorzugten Ausführungsform ist die Abrufsteuereinheit **3108** als eine herkömmliche (one-hot) codierte Zustandseinrichtung mit Flip-Flops in einer Gruppe von CL-Blöcken ausgeführt. Der Fachmann erkennt, daß in einer alternativen Ausführungsform die Abrufsteuereinheit **3108** auch als eine herkömmliche codierte Zustandseinrichtung oder als eine auf ROM basierende Zustandseinrichtung konfiguriert sein könnte. Der Befehlsspeicher **3110** schafft eine vorübergehende Speicherung für Befehle, die aus dem Speicher **3034** geladen sind. Für die Durchführung einer äußeren Schleifen-ISA ist der Befehlspuffer **3110** vorzugsweise als ein herkömmlicher, auf RAM basierender FIFO-Puffer mit einer Vielzahl von CL-Blöcken ausgeführt. Für die Ausführung einer Innenschleifen-ISA ist der Befehlspuffer **3110** vorzugsweise als ein Satz von Flip-Flop-Registern mit einer Anzahl Flip-Flops in einer Gruppe von IOBs oder einer Vielzahl Flip-Flops sowohl in IOBs als auch CLBs ausgeführt.

Die Decodiersteuereinheit **3112** richtet die Übertragung von Befehlen von dem Befehlspuffer **3110** in den Befehlsdecodierer **3114** entsprechend dem Decodiersignal aus, das von der ISS **3100** abgegeben worden ist. Für eine Innenschleifen-ISA ist die Decodiersteuereinheit **3112** vorzugsweise als eine auf ROM-basierende Zustandseinrichtung mit einem auf CLB basierenden ROM ausgeführt, der mit einem auf CLB basierenden Register verbunden ist. Für eine Außenschleifen-ISA ist die Decodiersteuereinheit **3112** als eine auf CLB basierende codierte Zustandseinrichtung ausgeführt. Für jeden Befehl, der als Eingangssignal empfangen wird, gibt der Befehlsdecoder **3114** einen entsprechenden Operationscode eine Register-Dateiadresse und wahlweise ein- oder mehrere Konstanten in herkömmlicher Weise ab. Für eine Innenschleifen-ISA ist der Befehlsdecodierer **3115** vorzugsweise entsprechend konfiguriert, um eine Gruppe von Befehlen, die als Eingangssignal empfangen worden sind, zu decodieren. In der bevorzugten Ausführungsform wird der Befehlsdecodierer **3114** als ein auf CLB-basierender Decodierer ausgeführt, der entsprechend konfiguriert ist, um jeden der Befehle, die in der aktuellen in Betracht gezogenen ISA enthalten sind, zu decodieren.

Der Operationscode-Speicheuegister-(OSR-)Satz **3113** schafft eine vorübergehende Speicherung für jeden Operationscodeausgang durch den Befehlsdecodierer **3144** und gibt jeden Operationscode an ISS **3100** ab. Wenn eine Außenschleifen-ISA in der DRP-Einheit **3032** durchgeführt wird, wird der OSR-Satz **3113** vorzugsweise mit Hilfe einer optimalen Anzahl von Flip-Flop-Registerbänken ausgeführt. Die Flip-Flop-Registerbänke erhalten Signale von dem Befehlsdecodierer **3114**, welche Klassen- oder Gruppencodes darstellen, die von Operationscode-Literal-Bitfeldern aus Befehlen abgeleitet sind, die vorher den Befehlspuffer **3110** in Form einer Warteschlange durchlaufen haben. Die Flip-Flop-Registerbänke speichern die vorerwähnten Klassen- oder Gruppencodes gemäß einem Decodierschema, das vorzugsweise eine ISS-Komplexität minimiert. Im Falle einer Innenschleifen-ISA speichert der OSR-Satz **3113** vorzugsweise Operationscode-Anzeigesignale, die unmittelbar von Operationscode-Literal-Bitfeldern abgeleitet worden sind, die von dem Befehlsdecodierer **3114** abgegeben worden sind. Innenschleifen-ISAs haben notwendigerweise kleinere Operationscode-Literal-Bitfelder, um dadurch die Ausführungsanforderungen hinsichtlich Puffern, Decodierern und einer Operationscode-Anzeige für ein sequentielles Befehlszuordnen durch den Befehlspuffer **3110**, den Befehlsdecodierer **3114** bzw. den OSR-Satz 3116 zu minimieren. Für Außenschleifen-ISAs wird der OSR-Satz **3116** vorzugsweise als ein kleiner Verbund von Flip-Flops-Registerbänken ausgeführt, die durch eine Bandbreite gekennzeichnet sind, die gleich der Operationscode-Literalgröße oder einem Bruchteil derer ist. Für Innenschleifen-ISAs ist der Operationscode-Speicherregister-(OSR-)Satz **3116** vorzugsweise eine kleinere und vereinheitlichte Flip-Flop-Registerbank als für Außenschleifen-ISAs. Die reduzierte Flip-Flop-Registerbankgröße im Fall einer Innenschleife reflektiert die minimale Befehls-Zählcharakteristik von Innenschleifen-ISAs relativ zu Außenschleifen-ISAs.

Der HF-Adressenregister-Satz **3118** und der Konstanten-Registersatz **3120** schaffen eine vorübergehende Speicherung für jede RegisterDateiadresse bzw. jeden konstanten Ausgang für den Befehlsdecodierer **3114**. In der bevorzugten Ausführungsform sind der Operationscode-Speicherregistersatz **3116**, der HF-Adressenregistersatz **3118** und der Konstanten-Registersatz **3112** jeweils als ein Satz von CL-Blöcken ausgeführt, die für eine Datenspeicherung konfiguriert sind.

Die Speicherzugriffslogik **3102** ist eine Speichersteuerschaltung, welche den Datentransfer zwischen dem Speicher **3034**, der DO-Einheit **3062** und der AO-Einheit **3064** entsprechend der Speicheradressengröße ausrichtet und synchronisiert, die in dem Architektur-Beschreibungsspeicher **3122** spezifiziert ist. Die Speicherzugriffslogik **3102** richtet aus und synchronisiert den Datentransfer und Befehle zwischen der S-Einrichtung **3012** und einer vorgegebenen T-Einrichtung **3014**. In der bevorzugten Ausführungsform stützt die Speicherzugriffslogik **3102** Stoßbetrieb-Speicherzugriffe und ist vorzugsweise als eine herkömmliche RAM-Steuereinheit mit CL-Blöcken ausgeführt. Der Fachmann erkennt, daß während einer Rekonfiguration die Eingangs- und Ausgangsanschlüsse der rekonfigurierbaren Logik für drei Zustände ausgelegt sind, so daß Widerstandsabschlüsse nicht bestehende Logikpegel definieren, und folglich nicht den Speicher **3034** stören. In einer anderen Ausführungsform könnte die Speicherzugriffslogik **3201** extern in der DRP-Einheit **3032** ausgeführt sein.

In **Fig.** 16 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer Datenoperations-(DO-)Einheit **3062** dargestellt. Die DO-Einheit **3062** führt Operationen an Daten gemäß DOU-Steuersignalen, HF-Adressen und Konstanten durch, die von ISS **3100** erhalten worden sind. Die DO-Einheit **3062** weist einen DOU-Crossbar-Schalter **3150**, eine Speicher/Ausrichtlogik **3152** und eine Datenoperationslogik **3154** auf. Der Crossbar-Schalter **3150**, die Speicher/Ausrichtlogik **3152** und die Datenoperationslogik **3154** haben jeweils einen Steuereingang, der mit dem ersten Steueraus-

gang der IF-Einheit **3060** über die erste Steuerleitung **3070** verbunden ist. Der Crossbar-Schalter **3150** hat einen zweigerichteten Dateneingang, welcher den zweigerichteten Dateneingang der DO-Einheit bildet, einen konstanten Eingang, der mit der dritten Steuerleitung **3074** verbunden ist, einen ersten Datenrückkopplungseingang, der mit einem Datenausgang der Datenoperationslogik **3154** über eine erste Datenleitung **3160** verbunden ist, einen zweiten Datenrückkopplungseingang, der mit einem Datenausgang der Speicher/Ausrichtlogik **3152** über eine zweite Datenleitung **3164** verbunden ist, und einen Datenausgang, der mit einem Dateneingang der Speicher/Ausrichtlogik **3152** über eine dritte Datenleitung **3162** verbunden ist. Zusätzlich zu dem Datenausgang hat die Speicher/Ausrichtlogik **3154** einen Adresseneingang, welcher mit der dritten Steuerleitung **3074** verbunden ist. Die Datenoperationslogik **3154** hat zusätzlich einen Dateneingang, welcher mit dem Ausgang der Speicher/Ausrichtlogik über die zweite Datenleitung **3164** verbunden ist.

Die Datenoperationslogik **3154** führt Rechen-, Schiebe- und/oder logische Operationen an Daten, welche an deren Dateneingang empfangen worden sind, entsprechend den DOU-Steuersignalen durch, die an deren Steuereingang empfangen worden sind. Die Speicher/Ausrichtlogik **3152** weist Datenspeicherelemente auf, die für eine vorübergehende Speicherung von Operanden, Konstanten und Teilergebnissen sorgen, die Datenberechnungen zugeordnet sind, und zwar unter der Anordnung von HF-Adressen und DOU-Steuersignalen, die an deren Adressen- bzw. Steuereingang empfangen worden sind. Der Crossbar-Schalter **3150** ist vorzugsweise ein herkömmliches Crossbar-Steuernetz, welches das Laden von Daten aus dem Speicher **3034**, dem Transfer von Ergebnissen, die von der Datenoperationslogik **3154** abgegeben worden sind, an die Speicher/Ausrichtlogik **3152** oder den Speicher **3034** und das Laden von Konstanten, die von der IF-Einheit **3060** abgegeben worden sind, in die Speicher/Ausrichtlogik **3152** entsprechend den DOU-Steuersignalen erleichtern, die an deren Steuereingang erhalten worden sind. In der bevorzugten Ausführungsform hängt der Aufbau der Datenoperationslogik **3154** im einzelnen von den Operationsarten ab, die von der aktuellen in Betracht gezogenen ISA gestützt sind. Das heißt, die Datenoperationslogik **154** weist eine Schaltung zum Durchführen der Rechen- und/oder logischen Operationen auf, welche durch die Datenoperationsbefehle in der aktuell in Betracht gezogenen ISA spezifiziert sind. Dementsprechend hängt auch der Aufbau der Speicher/Ausrichtlogik **3152** und des Crossbar-Schalters **3150** von der aktuell in Betracht gezogenen ISA ab. Der detaillierte Aufbau der Datenoperationslogik **3154**, die Speicher/Ausrichtlogik **3152** und der Crossbar-Speicher **3150** gemäß dem ISA-Typ wird nachstehend anhand der **Fig.** 17A bis 17B beschrieben.

Für eine Außenschleifen-ISA ist die DO-Einheit **3062** vorzugsweise entsprechend konfiguriert, um serielle Operationen an Daten durchzuführen. In **Fig.** 17A ist ein Blockdiagramm einer ersten beispielhaften Ausführungsform der DO-Einheit **3061** dargestellt, die für die Ausführung einer generellen Außenschleifen-ISA konfiguriert ist. Die generelle Außenschleifen-ISA erfordert Hardware, die konfiguriert ist, um mathematische Operationen, wie Multiplikationen, Additionen und Subtraktionen, Boolsche Operationen, wie UND, ODER und NOT, sowie Verschiebe- und Drehoperationen durchzuführen. Folglich weist für die Durchführung einer generellen Außenscheifen-ISA die Datenoperationslogik **5154** vorzugsweise eine Arithmetik-Logik-(AL-)Einheit/Schiebeeinheit **3184** mit einem ersten und zweiten Eingang, einem Steuereingang und einem Ausgang auf. Die Speicher/Ausrichtlogik **3152** weist vorzugsweise einen ersten RAM **3180** und einen zweiten RAM **3182** auf, die jeweils einen Dateneingang, einen Datenausgang, einen Adressen-Auswähleingang und einen Freigabeeingang haben. Der DOU-Crossbar-Schalter **3150** weist vorzugsweise ein herkömmliches Crossbar-Schaltnetz mit sowohl zweigerichteten als auch einseitig gerichteten Crossbar-Verbindungen und mit den vorstehend in Verbindung mit **Fig.** 16 beschriebenen Eingängen und Ausgängen auf. Der Fachmann weiß, daß eine effiziente Ausführung des DOU-Crossbar-Schalters **3150** für ein Außenschleifen-ISA Multiplexer, Puffer mit drei Zuständen, eine auf CLB basierende Logik, eine direkte Verdrahtung oder Untergruppen der vorerwähnten Elemente enthalten kann, die in irgendeine Kombination durch eine rekonfigurierbare Verbindungseinrichtung verbunden sind. Für dem Außenschleifen-ISA ist der DUO-Crossbar-Schalter **3150** entsprechend ausgeführt, um eine serielle Datenbewegung in einer minimal möglichen Zeit durchzuführen, während auch eine maximale Anzahl von eindeutigen Datenbewegungs-Crossbar-Verbindungen geschaffen wird, um verallgemeinerte Außenschleifen-Befehlstypen zu stützen.

Der Dateneingang des ersten und zweiten RAM **3180** bzw. **3182** ist mit dem Datenausgang des Schalter **3150** über die dritte Datenleitung **3162** verbunden. Die Adressenauswähleingänge der beiden RAM **3180** und **3182** sind entsprechend verbunden, um Registerdatei-Adressen von der IF-Einheit **3060** über die dritte Steuerleitung **3074** aufzunehmen. In ähnlicher Weise sind Freigabe-Eingänge der beiden RAM **3180, 3182** entsprechend verbunden, um DOU-Steuersignale über erste Steuerleitungen **3070** zu erhalten. Die Datenausgänge der beiden RAM **3180** und **3182** sind mit dem ersten und dem zweiten Eingang der ALU/Schiebeeinheit **3184** und auch mit dem zweiten Datenrückkopplungseingang des Schalters **3150** verbunden. Der Steuereingang der ALU-Schiebeeinheit **3184** ist entsprechend geschaltet, um DOU-Steuersignale über die erste Steuerleitung **3070** aufzunehmen, und der Ausgang der ALU-/Schiebeeinheit **3184** ist mit dem ersten Datenrückkopplungseingang des Schalters **3150** verbunden. Die Verbindungen zu den restlichen Eingängen und Ausgängen des Crossbar-Schalters **3150** sind identisch mit denen, die in Verbindung mit **Fig.** 16 vorstehend beschrieben worden sind.

Um das Ausführen eines Datenoperationsbefehls zu erleichtern, gibt die IF-Einheit **3060** DOU-Steuersignale, HF-Adressen- und Konstanten an die DO-Einheit **3061** während einer der ISS-Zustände E oder M ab. Die beiden RAM **3180, 3182** schaffen eine erste und zweite Registerdatei für ein vorübergehendes Datenspeichern. Einzelne Adressen in den beiden RAM **3180, 3182** werden entsprechend den HF-Adressen ausgewählt, die an jedem RAM-Adressenauswähleingang empfangen worden sind. Dementsprechend wird ein Laden der beiden RAM **3180, 3182** durch die DOU-Steuersignale gesteuert, die der jeweilige RAM **3180, 3182** an seinem Schreib-Freigabeeingang erhält. In der bevorzugten Ausführungsform weist zumindest ein RAM **3180, 3182** eine Durchgangsmöglichkeit auf, um den Datentransfer von dem DOU-Crossbar-Schalter **3150** unmittelbar in die ALU-Schiebeeinheit **3184** zu erleichtern. Die ALU/Schiebeeinheit **8134** führt arithmetische, logische oder Schiebeoperationen bei einem ersten Operanden, der von dem ersten RAM **3180** erhalten worden ist, und/oder bei einem zweiten Operanden, der von dem zweiten RAM **3182** erhalten worden ist, unter der Weisung der DOU-Steuersignale durch, die an deren Steuereingang erhalten worden sind. Der Crossbar-Schalter **3150** leitet selektiv 1) Daten zwischen dem Speicher **3034** und den beiden RAM **3180, 3182**, 2) Ergebnisse von der ALU/Schiebeeinheit **3184** zu den beiden RAM **3180, 3182** oder dem Speicher **3034**, 3) Daten, die in dem ersten oder zweiten

RAM **3180**, **3182** gespeichert sind, zu dem Speicher **3034**, und 4) Konstante von der IF-Einheit **3060** zu den beiden RAM **3180** und **3182**.

Wie vorstehend beschrieben, leitet für den Fall, daß entweder der erste oder der zweite RAM **3180**, **3182** eine Durchgangsmöglichkeit enthält, der Crossbar-Schalter **3150** selektiv auch Daten von dem Speicher **3034** oder von dem Ausgang der ALU/Schiebeeinheit unmittelbar zurück in die ALU-Schiebeeinheit **3184**. Der Crossbar-Schalter **3150** führt eine ganz bestimmte Schwenkoperation entsprechend den DOU-Steuersignalen durch, die an seinem Steuereingang empfangen worden sind. In einer bevorzugten Ausführungsform ist die ALU/Schiebeeinheit **3184** durch Verwenden von logischen Funktionsgeneratoren in einer Gruppe von LC-Blöcken und einer Schaltungsanordnung ausgeführt, die für mathematische Operationen in der rekonfigurierbaren Logikvorrichtung vorgesehen ist. Die beiden RAM **3180**, **3182** sind jeweils vorzugsweise mit Hilfe der Datenspeicherschaltung ausgeführt, die in einer Gruppe von CL-Blöcken vorhanden ist, und der Crossbar-Schalter **3150** ist vorzugsweise in der vorstehend beschriebenen Weise ausgeführt.

In **Fig.** 17B ist ein Blockdiagramm einer zweiten beispielshaften Ausführungsform einer DO-Einheit **3063** dargestellt, die für die Ausführung einer Innenschleifen-ISA konfiguriert ist. Im allgemeinen stützt eine innere Schleifen-ISA verhältnismäßig wenig spezialisierte Operationen und wird vorzugsweise dazu verwendet, eine gemeinsame Menge von Operationen an möglichst großen Datensätzen durchzuführen. Eine optimale Rechenleistung für eine Innenschleifen-ISA wird daher durch Hardware erzeugt, die entsprechend konfiguriert ist, um Operationen parallel durchzuführen. Folglich sind in der zweiten beispielhaften Ausführungsform der DO-Einheit **3063** die Datenoperationslogik **3154**, die Speicher/Ausrichtlogik **3152** und der DOU-Crossbar-Schalter **3150** entsprechend konfiguriert, um Pipeline-Berechnungen durchzuführen. Die Datenoperationslogik **3154** weist eine Pipeline-Funktionseinheit **3194** mit einer Anzahl Eingänge, einem Steuereingang und einem Ausgang auf. Die Speicher/Ausrichtlogik **3152** weist 1) eine Gruppe von herkömmlichen Flip-Flop-Anordnungen **3192**, wobei jede Flip-Flop-Anordnung **3132** einen Dateneingang und -ausgang sowie einen Steuereingang hat, und 2) einen Datenselektor **3190** auf, der einen Steuer- und einen Dateneingang sowie eine Anzahl Ausgänge hat, welche der Anzahl an vorhandenen Flip-Flops-Anordnungen **3192** entsprechen. Der DOU-Crossbar-Schalter **3150** weist ein herkömmliches Crossbar-Schaltnetz mit einseitig gerichteten Duplex-Crossbar-Verbindungen auf.

In der zweiten beispielhaften Ausführungsform der DO-Einheit **3063** enthält der Crossbar-Schalter **3150** vorzugsweise die Eingänge und Ausgänge, die vorher in Verbindung mit **Fig.** 16 beschrieben sind, mit Ausnahme des zweiten Datenrückkopplungseingangs. Analog zu dem Außenschleifen-ISA-Fall kann eine effiziente Ausführung des DOU-Crossbar-Schalters **3150** für eine Innenschleifen-ISA multipliziert, Puffer mit drei Zuständen, eine auf CLB-basierende Logik, eine direkte Verdrahtung oder eine Untergruppe der vorerwähnten Elemente enthalten, die in einer rekonfigurierbaren Weise verbunden sind. Über eine Innenschleifen-ISA ist der Schalter **3150** entsprechend ausgeführt, um parallele Datenbewegungen in einer minimalen Zeit zu maximieren, obwohl auch eine minimale Anzahl von eindeutigen Datenbewegungs-Crossbar-Verbindungen vorgesehen ist, um über eine Pipeline zugeführte Innenschleifen-ISA-Befehle zu stützen.

Der Dateneingang des Datenselektors **3190** ist mit dem Datenausgang des Schalters **3150** über die erste Datenleitung **3162** verbunden. Der Steuereingang des Datenselektors **3190** ist entsprechend verbunden, um HF-Adressen über die dritte Steuerleitung **3074** aufzunehmen, und jeder Ausgang des Datenselektors **3190** ist mit einem entsprechenden Dateneingang einer Flip-Flop-Anordnung verbunden. Der Steuereingang jeder Flip-Flop-Anordnung **3192** ist entsprechend geschaltet, um DOU-Steuersignale für die erste Steuerleitung **3070** aufzunehmen, und jeder Datenausgang der Flip-Flop-Anordnung ist mit einem Eingang der Funktionseinheit **3194** verbunden. Der Steuereingang der Funktionseinheit **3194** ist entsprechend geschaltet, um DOU-Steuersignale über die erste Steuerleitung **3070** zu erhalten, und der Ausgang der Funktionseinheit **3194** ist mit dem ersten Datenrückkopplungseingang des Crossbar-Schalters **3150** verbunden. Die Verbindungen der restlichen Eingänge und Ausgänge des Crossbar-Schalters **3150** sind identisch mit denjenigen, die vorher in Verbindung mit **Fig.** 16 beschrieben worden sind.

Während des Betriebs führt die Funktionseinheit **3194** Pipeline-Operationen an Daten, welche an deren Dateneingängen empfangen worden sind, entsprechend den DOU-Steuersignalen durch, die an deren Steuereingang erhalten worden sind. Der Fachmann erkennt, daß die Funktionseinheit **3194** eine Multiplizier-Addiereinheit, eine Schwellenwert-Bestimmungseinheit, eine Bilddreheinheit, eine Randvergrößerungseinheit oder eine Funktionseinheit ist, die sich dazu eignet, Pipeline-Operationen an aufgeteilten Daten durchzuführen. Der Datenselektor **3190** leitet Daten vom Ausgang des Schalters **3150** in eine vorgegebene Flip-Flop-Anordnung **3192** entsprechend den HF-Adressen, die an seinem Steuereingang empfangen worden sind. Jede Flip-Flop-Anordnung **3192** weist vorzugsweise eine Gruppe von fortlaufend verbundenen Datenhaltegliedern auf, um Daten relativ zu dem Dateninhalt der anderen Flip-Flop-Anordnung **3192** unter der Weisung der Steuersignale, die an deren Steuereingang erhalten worden sind, räumlich und vorübergehend auszurichten. Der Crossbar-Schalter **3150** leitet selektiv 1) Daten von dem Speicher **3034** zu dem Datenselektor **3190**, 2) Ergebnisse von der Multiplizier-Addiereinheit **3194** zu dem Datenselektor **3190** oder dem Speicher **3034** und 3) Konstante von der IF-Einheit **3060** zu dem Datenselektor **3190**. Der Fachmann erkennt, daß eine Innenschleifen-ISA eine Gruppe von "eingebauten" Konstanten haben kann. Bei der Durchführung einer solchen Innenschleifen-ISA enthält die Speicher-Ausrichtlogik **3154** vorzugsweise einen auf CLB basierenden ROM, welcher die "eingebauten" Konstanten enthält, um dadurch die Notwendigkeit zu beseitigen, Konstante von der IF-Einheit **3060** in die Speicher/Ausrichtlogik **3152** über den Schalter **3150** zu leiten. In der bevorzugten Ausführungsform ist die Funktionseinheit **3194** vorzugsweise unter Verwendung von logischen Funktionsgeneratoren und einer Schaltung ausgeführt, die für mathematische Operationen in einer Gruppe von CL-Blöcken vorgesehen ist. Jede Flip-Flop-Anordnung **3192** ist vorzugsweise unter Verwendung von Flip-Flops in einer Gruppe von CL-Blöcken ausgeführt, und der Datenselektor **3190** ist vorzugsweise unter Verwendung von logischen Funktionsgeneratoren und einer Datenauswählschaltung in einer Gruppe von CL-Blöcken ausgeführt. Schließlich ist die DOU-Crossbar-Schaltung **3150** vorzugsweise in der Weise ausgeführt, die vorher bezüglich der Innenschleifen-ISA beschrieben ist.

In **Fig.** 18 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer Adressenoperation-(AO-)Einheit **3064** dargestellt. Die AO-Einheit **3064** führt Operationen an Adressen entsprechend AOU-Steuersignalen, HF-Adressen und

Konstanten durch, die von der IF-Einheit **3060** erhalten worden sind. Die AO-Einheit **3064** weist einen AOU-Crossbar-Schalter **3200**, eine Speicher/Zähllogik **3202**, eine Adressen-Betriebslogik **3204** und einen Adressenmultiplexer **3206** auf. Der Schalter **3200**, die Speicher-Zähllogik **3202**, die Adressenbetriebslogik **3204** und der Adressenmultiplexer **3206** haben jeweils einen Steuereingang, der mit dem zweiten Steuereingang der IF-Einheit **3060** über die zweite Steuerleitung **3072** verbunden ist. Der Schalter **3200** hat einen zweigerichteten Datenzugang, welcher den zweigerichteten Datenzugang der AO-Einheit bildet, einen Adressenrückkopplungseingang, der mit einem Adressenausgang der Adressenoperationslogik **3204** über eine erste Adressenleitung **3210** verbunden ist, einen konstanten Eingang, der mit der dritten Steuerschaltung **3070** verbunden ist, und einen Adressenausgang, der mit einem Adresseneingang der Speicher/Zähllogik **3202** über eine zweite Adressenleitung **3212** verbunden ist. Außer dem Adressen- und Steuereingang hat die Speicher/Zähllogik **3202** einen H-Adresseneingang, der mit der dritten Steuerleitung **3074** verbunden ist, und einen Adressenausgang, der mit einem Adresseneingang der Adressenoperationslogik **3204** über eine dritte Adressenleitung **3214** verbunden ist. Der Adressenmultiplexer **3206** hat einen ersten Eingang, der mit der ersten Adressenleitung **3210** verbunden ist, einen zweiten Eingang, der mit der dritten Adressenleitung **3214** verbunden ist, und einen Ausgang, welcher dem Adressenausgang der AO-Einheit **3064** bildet.

Die Adressenoperationslogik **3204** führt Rechenoperationen an Adressen, die an deren Adresseneingang erhalten worden sind, unter der Anweisung von AOU-Steuersignalen durch, die an deren Steuereingang erhalten worden sind. Die Speicher/Zähllogik **3202** sorgt für eine vorübergehende Speicherung von Adressen und von Adressenrechenergebnissen. Der Crossbar-Schalter **3200** erleichtert das Laden von Adressen aus dem Speicher **3034**, den Transfer von Ergebnissen, die von der Adressenoperationslogik **3204** abgegeben worden sind, an die Speicher/Zähllogik **3202** oder den Speicher **3004** und das Laden von Konstanten, die von der IS-Einheit **3060** abgegeben worden sind, in die Speicher/Zähllogik **3202** entsprechend den AOU-Steuersignalen, die an deren Steuereingang erhalten worden sind. Der Adressenmultiplexer **3206** gibt selektiv eine Adresse, die von der Speicher/Zähllogik **3202** oder der Adressenoperationslogik **3200** erhalten worden ist, an den Adressenausgang der AO-Einheit **3064** unter der Anweisung der AOU-Steuersignale ab, die an deren Steuereingang empfangen worden sind. In der bevorzugten Ausführungsform ist der detaillierte Aufbau des AOU-Crossbar-Schalters **3200**, der Speicher/Ausrichtlogik **3202** und der Adressenbetriebseinheit **3204** abhängig von der Art der gerade in Betracht gezogenen ISA, was nachstehend unter Bezugnahme auf **Fig.** 19A und 19B beschrieben wird.

In **Fig.** 19A ist ein Blockdiagramm einer ersten beispielhaften Ausführungsform der AO-Einheit **6065** dargestellt, die für die Durchführung einer generellen Außenschleifen-ISA konfiguriert ist. Eine generelle Außenschleifen-ISA erfordert Hardware zum Durchführen von Operationen, wie einer Addition, einer Subtraktion, ein Inkrementieren und Dekrementieren bei dem Inhalt eines Programmzählers und von Adressen, die in der Speicher/Zähllogik **3202** gespeichert sind. In der ersten beispielhaften Ausführungsform der AO-Einheit **3065** weist die Adressenoperationslogik **3205** vorzugsweise ein das nächste Befehlsprogramm betreffendes Adressenregister (NIPAR) **3232** mit einem Eingang, einem Ausgang und einem Steuereingang, eine Recheneinheit **3234** mit ersten bis dritten Eingängen, einem Steuereingang und einem Ausgang und einen Multiplexer **3230** mit einem ersten und einem zweiten Eingang, einem Steuereingang und einem Ausgang auf. Die Steuer/Zähllogik **3202** weist vorzugsweise einen dritten RAM **3220** und einen vierten RAM **3222** auf, die jeweils einen Eingang, einen Ausgang, einen Adressenwähleingang und einen Freigabeeingang haben. Der Adressenmultiplexer **3206** weist vorzugsweise einen Multiplexer mit einem ersten, zweiten und dritten Eingang, einen Steuereingang und einem Ausgang auf. Der Crossbar-Schalter **3200** weist vorzugsweise einen herkömmliches Crossbar-Schaltnetz mit einseitig gerichteten Duplex-Crossbar-Verbindungen und mit den Eingängen und Ausgängen auf, die vorher unter Bezugnahme auf **Fig.** 19 beschrieben worden sind. Eine effektive Ausführung des AOU-Crossbar-Schalters **3200** kann Multiplexer, Puffer mit drei Zuständen, eine auf CLB-basierende Logik, eine direkte Verdrahtung und eine Untergruppe solcher Elemente aufweisen, die durch rekonfigurierbare Verbindungen verbunden sind. Für eine Außenschleifen-ISA ist der Crossbar-Schalter **3200** vorzugsweise ausgeführt, um eine serielle Adressenbewegung in einer minimalen Zeit zu maximieren, während auch eine maximale Anzahl eindeutiger Adressenbewegungs-Crossbar-Verbindungen vorgesehen ist, um verallgemeinerte eine Außenschleifen-ISA betreffende Pressenoperationsbefehle zu stützen.

Der Eingang des dritten und vierten RAM **3220**, **3222** ist jeweils mit dem Ausgang des Crossbar-Schalters **3200** über die zweite Adressenleitung **3212** verbunden. Die Adressenwähleingänge der dritten und vierten RAM **3220**, **3222** sind entsprechend geschaltet, um HF-Adressen von der IF-Einheit **3060** über die dritte Steuerleitung **74** aufzunehmen und die Freigabeeinänge der ersten und zweiten RAM **3220**, **3222** sind entsprechend geschaltet,um AOU-Steuersignale über die zweite Steuerleitung **3072** aufzunehmen. Der Ausgang des dritten RAM **3220** ist verbunden mit dem ersten Eingang des Multiplexers **3230**, mit dem ersten Eingang der Recheneinheit **3234** und dem ersten Eingang des Adressenmultiplexers **3206**. In ähnlicher Weise ist der Ausgang des vierten RAM **3222** verbunden mit dem zweiten Eingang des Multiplexers **3220**, dem zweiten Eingang der Recheneinheit **3234** und dem zweiten Eingang des Adressenmultiplexers **3206**. Die Steuereingänge des Multiplexers **3230**, des NIPAR-Registers **3232** und die Recheneinheit **3234** sind jeweils mit der zweiten Steuerleitung **3272** verbunden. Der Ausgang der Recheneinheit **3234** bildet den Ausgang der Adressenoperationslogik **3204** und ist folglich mit dem Adressenrückkopplungseingang des AOU-Crossbar-Schalters **3200** und dem dritten Eingang des Adressenmultiplexers **3206** verbunden. Die Verbindungen zu den restlichen Eingängen und Ausgängen des AOU-Crossbar-Schalters **3200** und des Adressenmultiplexers **3206** sind identisch mit denjenigen, die vorher unter Bezugnahme auf **Fig.** 18 beschrieben worden sind.

Um die Durchführung eines Adressen-Operationsbefehls zu vereinfachen, gibt die IF-Einheit **3060** AOU-Steuersignale, HF-Adressen und Konstanten an die AO-Einheit **3064** während entweder des ISS-Zustandes E oder M ab. Die dritten und vierten RAM **3220**, **3222** schaffen eine erste bzw. zweite Registerdatei für ein vorübergehendes Adressenspeichern. Einzelne Speicherstellen in dem dritten und vierten RAM **3220**, **3222** werden entsprechend dem HF-Adressen ausgewählt, die an jedem RAM-Adressenauswähleingang erhalten worden sind. Das Laden des dritten und vierten RAM **3220**, **3222** wird durch die AOU-Steuersignale gesteuert, die der jeweilige RAM **3220**, **3222** an seinem Schreib-Freigabeeingang erhält. Der Multiplexer **3230** leitet selektiv Adressen, die von dem dritten und vierten RAM **3220**, **3222** abgegeben worden sind, zu dem NIPAR-Register **3232** unter der Anweisung der AOU-Steuersignale, die an dessen Steuereingang erhalten worden sind. Das NIPAR-Register **3232** lädt eine Adresse, die von dem Ausgang des Multiplexers **3230**

erhalten worden ist, und inkrementiert deren Inhalt entsprechend den AOU-Steuersignalen, die an dessen Steuereingang erhalten worden sind.

In der bevorzugten Ausführungsform speichert das NIPAR-Register **3232** die Adresse des nächsten durchzuführenden Programmbefehls. Die Recheneinheit **3234** führt Rechenoperationen einschließlich einer Addition, einer Subtraktion, eines Inkrementierens und Dekrementierens an Adressen, die von dem dritten und vierten RAM **3220**, **3222** und/oder an Inhalten eines NIPAR-Registers **3232** durch. Der Schalter **3200** leitet selektiv 1) Adressen von dem Speicher **3034** zu dem dritten und vierten RAM **3220**, **3222** und 2) Ergebnisse von Adressenrechnungen, die von der Recheneinheit **3234** abgegeben worden sind, zu dem Speicher **3034** oder dem dritten und vierten RAM **3220**, **3222**. Der AOU-Crossbar-Schalter **3200** führt eine ganz bestimmte Lenkoperation gemäß dem AOU-Steuersignalen durch, die an dessen Steuereingang erhalten worden sind. Der Adressenmultiplexer **3206** lenkt selektiv Adressen, die von dem dritten RAM **3220** abgegeben worden sind, Adressen, die von dem vierten RAM **3222** abgegeben worden sind, oder Ergebnisse von Adressenrechnungen, die von der Recheneinheit **3234** abgegeben worden sind, zu dem AOU-Adressenausgang unter der Anweisung der AOU-Steuersignale, die an dessen Steuereingang erhalten worden sind.

In der bevorzugten Ausführungsform sind der dritte und vierte RAM **3220**, **3222** jeweils durch Verwenden der Datenspeicherschaltung ausgeführt, die in einem Satz von CL-Blöcken vorhanden ist. Der Multiplexer **3230** und der Adressenmultiplexer **3206** sind jeweils durch Verwenden einer Datenauswahlschaltung ausgeführt, die in einem Satz von CL-Blöcken vorhanden ist, und das NIPAR-Register **3232** ist vorzugsweise durch Verwenden einer Datenspeicherschaltung ausgeführt, die in einem Satz CL-Blöcken vorhanden ist. Die Recheneinheit **3234** ist vorzugsweise mit Hilfe von logischen Funktionsgeneratoren und einer Schaltung ausgeführt, die mathematischen Funktionen in einer Gruppe von CL-Blöcken zugeordnet ist. Schließlich ist der AOU-Crossbar-Schalter vorzugsweise in der vorstehend beschriebenen Weise ausgeführt.

In **Fig.** 19B ist ein Blockdiagramm einer zweiten beispielhaften Ausführungsform der AO-Einheit **3066** dargestellt, die für die Durchführung einer Innenschleifen-ISA konfiguriert ist. Vorzugsweise erfordert eine Innenschleifen-ISA Hardware zum Durchführen einer sehr begrenzten Menge von Adressenoperationen, und eine Hardware, um zumindest einen Quellenadressenzeiger und eine entsprechende Anzahl von Bestimmungsadressenzeigern aufrechtzuerhalten. Eine Innenschleifen-Verarbeitung, für welche eine sehr begrenzte Anzahl von Adressenoperationen oder sogar eine einzige Adressenoperation erforderlich sind, enthält Block- oder Serpentinenoperationen an Bilddaten, Bitumkehroperationen, Operationen an zirkularen Pufferdaten und Parsing-Operationen an Daten variabler Länge.

Nachstehend wird eine einzige Adressenoperation betrachtet, nämlich eine Inkrement-Operation. Der Fachmann erkennt, daß Hardware, welche Inkrement-Operationen durchführt, auch inhärent in der Lage ist, Dekrementier-Operationen durchzuführen, um dadurch eine zusätzliche Adressen-Operationsmöglichkeit zu schaffen. In der zweiten beispielhaften Ausführungsform der AO-Einheit **3076** weist die Speicher/Zähllogik **3202** zumindest auf ein Quellenadressenregister **3252** mit einem Eingang, einem Ausgang und einem Steuereingang, zumindest ein Bestimmungsadressenregister **3254** mit einem Eingang, einem Ausgang und einem Steuereingang und einen Datenselektor **3250** mit einem Eingang, einem Steuereingang und einer Anzahl Ausgänge, welche gleich der Gesamtanzahl an vorhandenen Quellen-Bestimmungs-Adressenregister **3252**, **3254** ist. Hier werden ein einziges Quellenadressenregister **3252** und ein einziges Bestimmungsregister **3254** betrachtet und daher hat der Datenselektor **3250** einen ersten und einen zweiten Ausgang.

Die Adressenoperations-Logik **3204** weist ein NIPAR-Register **3232** mit einem Eingang, einem Ausgang und einem Steuerausgang, wie einen Multiplexer **3260** auf, der eine Anzahl Eingänge, die gleich der Anzahl Datenselektorausgänge sind, einen Steuereingang und einen Ausgang hat. Hierbei hat der Multiplexer **3260** einen ersten Eingang und einen zweiten Eingang. Der Adressenmultiplexer **3206** weist vorzugsweise einen Multiplexer mit einer Anzahl Eingänge auf, welche um eins größer ist als die Anzahl an Datenselektorausgängen, einem Steuereingang und einem Ausgang auf. Folglich hat hier der Adressenmultiplexer **3201** einen ersten, einen zweiten und einen dritten Eingang. Der Crossbar-Schalter **3200** weist vorzugsweise ein herkömmliches Crossbar-Schaltnetz mit zweiseitig und einseitig gerichteten Crossbar-Verbindungen und mit den Eingängen und Ausgängen, die vorher in Verbindung mit **Fig.** 18 beschrieben worden sind. Eine effiziente Ausführung des AOU-Crossbar-Schalters **3200** kann enthalten Multiplexer, Puffer mit drei Zuständen, eine auf CLB basierende Logik, eine Direktverdrahtung oder eine Untergruppe solcher Elemente, die durch rekonfigurierbare Kopplungen verbunden sind. Für eine Innenschleifen-ISA ist der Schalter **3200** vorzugsweise ausgelegt, um eine parallele Adressenbewegung in einer minimal möglichen Zeit durchzuführen, während auch eine minimale Anzahl von eindeutigen Adressenbewegungs-Crossbar-Kopplungen vorgesehen sind, um Innenschleifen-Operationen zu stützen. Der Eingang des Datenselektors **3250** ist mit dem Ausgang des Schalters **3200** verbunden. Die ersten und zweiten Ausgänge des Datenselektors **3250** sind mit dem Eingang des Feldadressenregister **3252** bzw. des Bestimmungsadressen-Registers **3254** verbunden. Die Steuereingänge des Quellenadressenregisters **3252** und des Bestimmungsadressenregister **3254** erhalten AO-Steuersignale über die zweite Steuerleitung **3072**. Der Ausgang des Quellenadressenregisters **3252** ist mit dem ersten Eingang des Multiplexers **3260** und dem ersten Eingang des Adressenmultiplexers **3206** verbunden. Dementsprechend ist der Ausgang des Bestimmungsregisters **3254** mit dem zweiten Eingang des Multiplexers **3260** bzw. des Adressenmultiplexers **3206** verbunden. Der Eingang des NIPAR-Registers **3232** ist mit dem Ausgang des Multiplexers **3260** verbunden. Der Steuereingang des Registers **3232** erhält AOU-Steuersignale über die zweite Steuerleitung **3072**, und der Ausgang des Registers **3232** ist sowohl mit dem Adressenrückkopplungseingang des Schalters **3200** als auch mit dem dritten Eingang des Adressenmultiplexers **3206** verbunden. Die Verbindungen zu den restlichen Eingängen und Ausgängen des Schalters **3200** sind identisch mit dem, die vorher bezüglich **Fig.** 18 beschrieben worden sind.

Beim Betrieb leitet der Datenselektor **3250** Adressen, die von dem AOU-Crossbar-Schalter **3200** erhalten worden sind, zu dem Quellenadressenregister **3252** oder dem Bestimmungsadressenregister **3254** entsprechend den HF-Adressen, die an dem Steuereingang erhalten worden sind. Das Quellenadressenregister **3252** lädt eine an dessen Eingang vorhandene Adresse entsprechend den an dessen Steuereingang vorhandenen AOU-Steuersignale. Das Bestimmungsadressenregister **3254** lädt eine an dessen Eingang vorhandene Adresse in analoger Weise. Der Multiplexer **3260** leitet eine Adresse, die er von dem Quellenadressenregister **3252** oder dem Bestimmungsadressenregister **3254** erhalten hat, zu

dem Eingang des NIPAR-Registers **3232** entsprechend der AOU-Steuersignalen, die er an dem Steuereingang erhalten hat. Das Register **3232** lädt eine an dessen Eingang vorhandene Adresse, inkrementiert oder dekrementiert dessen Inhalt entsprechend den an dem Steuereingang erhaltenen AOU-Steuersignalen. Der AOU-Crossbar-Schalter **3200** lenkt selektiv 1) Adressen von dem Speicher **3034** zu dem Datenselektor **3050** und 2) den Inhalt des Registers **3232** zu dem Speicher **3034** oder dem Datenselektor **3250**. Der AOU-Crossbar-Schalter **3200** führt eine ganz bestimmte Leitoperation entsprechend den an dem Steuereingang erhaltenen AOU-Steuersignalen. Der Adressenmultiplexer **3206** leitet selektiv den Inhalt des Quellenadressenregisters **3252**, des Bestimmungsadressenregisters **3254** oder des NIPAR-Registers **3232** zu dem AOU-Adressenausgang unter der Anweisung der an dessen Steuereingang erhaltenen AOU-Steuersignalen.

In der bevorzugten Ausführungsform sind das Quellenadressenregister **3252** und das Bestimmungsadressenregister **3254** jeweils unter Verwendung der Datenspeicherschaltung ausgeführt, die in einem Satz von CL-Blöcken vorhanden ist. Das NIPAR-Register **3232** ist vorzugsweise mit Hilfe einer Inkrementier/Dekrementier-Logik und Flip-Flops in einem Satz von CL-Blöcken ausgeführt. Der Datenselektor **3250**, der Multiplexer **3220** und der Adressenmultiplexer **3206** sind jeweils vorzugsweise unter Verwendung einer Datenselektionsschaltung ausgeführt, die in einem Satz CL-Blöcken vorhanden ist. Schließlich ist der Schalter **3200** vorzugsweise in der Weise ausgeführt, die vorher für eine Innenschleifen-ISA beschrieben ist. Der Fachmann erkennt, daß in bestimmten Anwendungen es vorteilhaft sein kann, eine ISA zu benutzen, welche auf einer Innenschleifen-AOU-Konfiguration einer Außenschleifen-DOU-Konfiguration oder umgekehrt beruht. Beispielsweise würde eine assoziative String-Suchvorgang-ISA in vorteilhafter Weise eine Innenschleifen-DOU-Konfiguration zusammen mit einer Außenschleifen-AOU-Konfiguration ausnutzen. Ferner würde beispielsweise eine ISA zum Durchführen von Histogramm-Operationen in vorteilhafter Weise eine Außenschleifen-DOU-Konfiguration in Verbindung mit einer Innenschleifen-AOU-Konfiguration ausnutzen.

Begrenzte rekonfigurierbare Hardware-Ressourcen müssen zwischen jedem Element der DRPU-Einheit **3032** angeordnet werden. Da die rekonfigurierbaren Hardware-Ressourcen in ihrer Zahl begrenzt sind, beeinflußt die Art und Weise, in welcher sie der IF-Einheit **3060** beispielsweise zugeordnet werden, den maximalen Rechenleistungspegel, der von der DA-Einheit **3062** und der AO-Einheit **3064** erreichbar ist. Die Art und Weise, in welcher die rekonfigurierbaren Hardware-Ressourcen zwischen der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** zugeordnet werden, ändert sich entsprechend dem Typ von ISA, der in einem vorgegebenen Moment durchzuführen ist. Wenn die ISA-Komplexität zunimmt, müssen mehrere konfigurierbare Hardware-Ressourcen der IF-Einheit **3060** zugeordnet werden, um zunehmend komplexe Decodier- und Steueroperationen zu vereinfachen, die weniger rekonfigurierbare Hardware-Ressourcen übriglassen, die zwischen der DO-Einheit **3063** und der AO-Einheit **3064** verfügbar sind. Folglich nimmt die maximale Rechenleistung, die von der DO-Einheit **3062** und der AO-Einheit **3064** erreichbar ist, mit der ISA-Komplexität ab. Im allgemeinen hat eine Außenschleifen-ISA viel mehr Befehle als eine Innenschleifen-ISA, und daher ist deren Ausführung merklich komplexer hinsichtlich der Decodier- und Steuerschaltung. Beispielsweise würde eine Außenschleifen-ISA, die einen 64 Bit Universalprozessor definiert, viel mehr Befehle haben als eine Innenschleifen-ISA, die nur auf eine Datenkompression ausgerichtet ist.

In **Fig.** 20A ist ein Diagramm dargestellt, das eine beispielhafte Zuordnung von rekonfigurierbaren Hardware-Ressourcen zwischen IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** für eine Außenschleifen-ISA zeigt. In der beispielhaften Zuordnung der rekonfigurierbaren Hardware-Ressourcen für die Außenschleifen-ISA sind der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** jeweils annähernd ein Drittel der verfügbaren rekonfigurierbaren Hardware-Ressourcen zugeordnet. Für den Fall, daß die DRP-Einheit **3032** zu rekonfigurieren ist, um eine Innenschleifen-ISA durchzuführen, werden weniger rekonfigurierbare Hardware-Ressourcen erfordert, um die IF-Einheit **3060** und die AO-Einheit **3064** infolge der begrenzten Anzahl von Befehlen und Arten von Adressenoperationen auszuführen, die von einer Innenschleifen-ISA getragen sind.

In **Fig.** 20B ist ein Diagramm dargestellt, das eine beispielhafte Zuordnung von rekonfigurierbaren Hardware-Ressourcen zwischen der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** für eine Innenschleifen-ISA zeigt. In der beispielhaften Zuordnung von rekonfigurierbaren Hardware-Ressourcen für die Innenschleifen-ISA ist die IF-Einheit **3060** unter Verwendung von annähernd 5 bis 10% der rekonfigurierbaren Hardware-Ressourcen ausgeführt, und die AO-Einheit **3064** ist unter Verwendung von annähernd 10 bis 25% der rekonfigurierbaren Hardware-Ressourcen ausgeführt. Folglich bleiben etwa 70 bis 80% der rekonfigurierbaren Hardware-Ressourcen verfügbar, um die DO-Einheit **3062** auszuführen. Dies wiederum bedeutet, daß der innere Aufbau der DO-Einheit **3062**, welche der Innenschleifen-ISA zugeordnet ist, komplexer sein kann und folglich ein deutlich höheres Leistungsvermögen bietet als der interne Aufbau der DO-Einheit **3062**, welche der Außenschleifen-ISA zugeordnet ist.

Der Fachmann erkennt, daß die DRP-Einheit **3032** entweder die DO-Einheit **3062** oder die AO-Einheit **3064** in einer alternativen Ausführungsform enthalten kann. Beispielsweise braucht in einer alternativen Ausführungsform die DRP-Einheit **3032** nicht eine AO-Einheit **3064** zu enthalten. Die DO-Einheit **3062** würde dann dafür verantwortlich sein, Operationen sowohl bei Daten als auch bei Adressen durchzuführen. Ungeachtet der besonderen, in Betracht gezogenen DRPU-Ausführungsform muß eine endliche Anzahl rekonfigurierbarer Hardware-Ressourcen zugeordnet werden, um die Elemente der DRPU-Einheit **3032** auszuführen. Die rekonfigurierbaren Hardware-Ressourcen sind vorzugsweise so zugeordnet, daß eine optimale oder beinahe optimale Leistung für die aktuell in Betracht gezogene ISA relativ zu dem Gesamtraum an verfügbaren rekonfigurierbaren Hardware-Ressourcen erreicht wird.

Der Fachmann erkennt, daß der detaillierte Aufbau des Elements der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** nicht auf die vorstehend beschriebenen Ausführungsformen beschränkt ist. Für eine vorgegebene ISA ist der entsprechende Konfigurations-Datensatz vorzugsweise so definiert, daß der interne Aufbau jedes Elements in der IF-Einheit **3060**, der DO-Einheit **3062** und der AO-Einheit **3064** eine Rechenleistung relativ zu den verfügbaren rekonfigurierbaren Hardware-Ressourcen maximiert.

In **Fig.** 21 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer T-Einrichtung **3014** dargestellt. Die T-Einrichtung **3014** weist eine zweite lokale Zeitbasiseinheit **3300**, eine gemeinsame Interface- und Steuereinheit **3302** und eine Reihe Verbindungs-Ein/Ausgabeeinheiten **3304** auf. Die zweite lokale Zeitbasiseinheit **3300** hat einen Zeitsteuereingang, welcher den Master-Zeitsteuereingang der T-Einrichtung bildet. Die gemeinsame Interface-Steuereinheit **3302**

hat einen Zeitsteuereingang, der mit einem Zeitsteuereingang der zweiten lokalen Zeitbasiseinheit **3300** über eine zweite Zeitsteuersignalleitung **3310** verbunden ist, einen Adressenausgang, welcher mit der Adressenleitung **3344** verbunden ist, einen ersten zweigerichteten Dateneingang, der mit der Speicher-Ein/Ausgabe-Leitung **3044** verbunden ist, einen zweigerichteten Steuereingang, der mit einer externen Steuerleitung **3048** verbunden ist, und einen zweiten zweigerichteten Dateneingang, der mit einem zweigerichteten Dateneingang jeder vorhandenen Ein/Ausgabeeinheit **3304** über eine Nachrichtenübertragleitung **3312** verbunden ist. Jede Ein/Ausgabeeinheit **3304** hat einen Eingang, welcher mit der GPI-Matrix **3016** über eine Nachrichteneingangsleistung **3314** verbunden ist, und einen Ausgang, der mit der GPI-Matrix **3016** über eine Nachrichtenausgangsleitung **3316** verbunden ist.

Die zweite lokale Zeitbasis-Einheit **3300** in der T-Einrichtung **3014** erhält das Master-Zeitsteuersignal von der Master-Zeitbasiseinheit **3022** und erzeugt ein zweites lokales Zeitsteuersignal. Die zweite lokale Zeitbasiseinheit **3300** liefert das zweite lokale Zeitsteuersignal an die gemeinsame Interface-Steuereinheit **3302**, um dadurch eine Zeitsteuerreferenz für die T-Einrichtung **3040** zu schaffen, in welcher sie untergebracht ist. Vorzugsweise ist die zweite lokale Zeitsteuereinheit phasensynchronisiert mit dem Master-Zeitsteuersignal. In dem System **1030** arbeitet jede zweite lokale Zeitbasiseinheit **3300** der T-Einrichtung vorzugsweise auf derselben Frequenz. Der Fachmann erkennt, daß in einer alternativen Ausführungsform ein oder mehrere zweite lokale Zeitbasiseinheiten **3300** auch auf unterschiedlichen Frequenzen arbeiten können. Die zweite lokale Zeitbasiseinheit **3300** ist vorzugsweise mit Hilfe einer herkömmlichen phasenstarren Frequenzumwandlungsschaltung ausgeführt, die eine auf CLB basierende phasenstarre Detektionsschaltung enthält. Der Fachmann erkennt, daß in einer alternativen Ausführungsform die zweite lokale Zeitbasiseinheit **3300** als ein Teil eines Taktverteilungsbaums ausgeführt sein könnte.

Die gemeinsame Interface-Steuereinheit **3302** leitet den Nachrichtentransfer zwischen der entsprechenden S-Einrichtung **3012** und einer spezifizierten Ein-/Ausgabeeinheit **3304**, wobei eine Nachricht einen Befehl und möglicherweise Daten enthält. In der bevorzugten Ausführungsform kann die spezifizierte Ein/Ausgabeeinheit **3304** in einer T-Einrichtung **3014** oder in einer Ein/Ausgabe-T-Einrichtung **3018** intern oder extern zu dem System **3010** untergebracht sein. In der vorliegenden Erfindung ist jede Ein/Ausgabeeinheit **3304** vorzugsweise einer Verbindungsadresse zugeordnet, die eindeutig die verbindende Ein/Ausgabeeinheit **3304** identifiziert. Die Verbindungsadressen für die Ein/Ausgabeeinheiten **3304** in einer vorgegebenen T-Einrichtung werden in dem entsprechenden Architektur-Beschreibungsspeicher **3101** der S-Einrichtung gespeichert.

Die gemeinsame Interface- und Steuereinheit **3302** erhält Daten und Befehle von der entsprechenden S-Einrichtung **3012** über die Speicher-Ein-/Ausgabeleitung **3064** bzw. die externe Steuersignalleitung **3048**. Vorzugsweise enthält jeder empfangene Befehl eine Sollkopplungsadresse und einen Befehlscode, welcher eine ganz bestimmte durchzuführende Operationsart spezifiziert. In der bevorzugten Ausführungsform enthalten die Operationsarten, die eindeutig durch Befehlscodes codiert sind, 1) Datenleseoperationen, 2) Datenschreiboperationen und 3) einen Unterbrechungssignal-Transfer einschließlich eines Rekonfigurations- Unterbrechungstransfers. Die Sollkopplungsadresse identifiziert eine Sollkopplungs-Ein/Ausgabeeinheit **3304**, an welcher Daten und Befehle zu übertragen sind. Vorzugsweise überträgt die gemeinsame Interface-Steuereinheit **3302** jeden Befehl und irgendwelche diesbezüglichen Daten als einen Satz auf einem Datenpaket basierender Nachrichten in herkömmlicher Weise, wobei jede Nachricht die Sollkopplungsadresse und den Befehlscode enthält.

Außer Daten und Befehle von der entsprechenden S-Einrichtung **3012** zu erhalten, erhält die gemeinsame Interface- und Steuereinheit **3302** Nachrichten von jeder der Ein/Ausgabeeinheit **3304**, die mit der Nachrichtenübertragungsleitung **3312** verbunden sind. In der bevorzugten Ausführungsform wandelt die Interface- und Steuereinheit **3302** eine Gruppe von diesbezüglichen Nachrichten in eine einzige Befehls- und Datenfolge um. Wenn der Befehl an die DRP-Einheit **3032** in der entsprechenden S-Einrichtung **3012** gerichtet ist, gibt die Interface- und Steuereinheit **3302** den Befehl über die externe Steuersignalleitung **3048** ab. Wenn der Befehl an den Speicher **3034** in der entsprechenden S-Einrichtung **3012** gerichtet ist, gibt die Interface- und Steuereinheit **3302** ein entsprechendes Speichersteuersignal über die externe Steuersignalleitung **3048** und ein Speicheradressensignal über die Speicheradressenleitung **3044** ab. Daten werden über die Speicher-Ein/Ausgabeeinheit **3046** übertragen. In der bevorzugten Ausführungsform weist die Interface- und Steuereinheit **3302** eine auf CLB basierende Schaltung auf, um Operationen analog denjenigen auszuführen, die von einer herkömmlichen SCI-Schalteinheit ausgeführt worden sind, die durch ANSI-/IEEE-Norm 1596 bis 1992 definiert ist.

Jede Ein/Ausgabeeinheit **3304** erhält Nachrichten von der Interface- und Steuereinheit **3302** und überträgt Nachrichten an andere Ein/Ausgabeeinheiten **3304** über die GPI-Matrix **3016** unter Anweisung von Steuersignalen, die von der Interface- und Steuereinheit **3302** erhalten worden sind. In der bevorzugten Ausführungsform basiert die Ein/Ausgabeeinheit **3304** auf einem SCI-Knoten, wie er durch ASNI/IEEE-Norm 1596–1992 definiert ist. In **Fig.** 22 ist ein Blockdiagramm einer bevorzugten Ausführungsform einer Kopplungs-Ein/Ausgabeeinheit **3304** dargestellt. Die Einheit **3304** weist einen Adressendecodierer **3320**, einen Eingabe-FIFO-Puffer **3322**, einen Bypass-FIFO-Puffer **3324**, einen Ausgangs-FIFO **3326**, und einen Multiplexer **3328** auf. Der Adressendecodierer **3320** hat einen Eingang, welcher den Eingang einer Kopplungs-Ein/Ausgabeeinheit **3304** bildet, einen ersten Ausgang, der mit dem Eingabe-FIFO **3322** verbunden ist, und einen zweiten Ausgang, der mit dem Bypass-FIFO **3324** verbunden ist. Das Eingabe-FIFO **3322** hat einen Ausgang, welcher mit der Nachrichtenübertragungsleitung **3312** verbunden ist, um Nachrichten an die gemeinsame Interface- und Steuereinheit **3302** zu übertragen. Das Ausgangs-FIFO **3326** hat einen Eingang, welcher mit der Nachrichtenübertragungsleitung **2312** verbunden ist, um Nachrichten von dem gemeinsamen Interface und einer Steuerschaltung **3302** zu erhalten, und einen Ausgang, der mit einem ersten Eingang des Multiplexers **3328** verbunden ist. Der Bypass-FIFO **3326** hat einen Ausgang, der mit einem zweiten Eingang des Multiplexers **3328** verbunden ist. Schließlich hat der Multiplexer **3328** einen Steuereingang, welcher mit der Nachrichtenübertragungsleitung **3312** verbunden ist, und einen Ausgang, der den Ausgang der Kopplungs-Ein/Ausgabeeinheit bildet.

Die Ein/Ausgabeeinheit **3304** erhält Nachrichten an dem Eingang des Adressen-Decodierers **3312**, welcher bestimmt, ob die Sollkopplungsadresse, die in der empfangenen Nachricht spezifiziert ist, identisch mit der Kopplungsadresse der Kopplungs-Ein/Ausgabeeinheit **3304** ist, in welcher sie untergebracht ist. Wenn dem so ist, leitet der Adressendecodierer **3320** die Nachricht zu dem Eingabe-FIFO **3322**. Anderenfalls leitet der Adressendecodierer **3320** die Nachricht zu dem

Bypass-FIFO **3324**. In der bevorzugten Ausführungsform weist der Adressen-Decodierer **3320** einen Codierer und einen Datenselektor auf, die mit Hilfe IOBs und CLBs ausgeführt sind.

In **Fig.** 23 ist ein Blockdiagramm einer bevorzugten Ausführungsform der Ein/Ausgabe-T-Einrichtung **3018** dargestellt. Die Ein/Ausgabe-T-Einrichtung **3018** weist eine dritte lokale Zeitbasiseinheit **3360**, eine gemeinsame kundenspezifische Interface- und Steuereinheit **3362** und eine Kopplungs-Ein/Ausgabeeinheit **3304** auf. Die dritte lokale Zeitbasiseinheit **3360** hat einen Zeitsteuereingang, welche den Master-Zeitsteuereingang der Ein/Ausgabe-T-Einrichtung **3018** bildet. Die Ein/Ausgabe-Einheit **3304** hat einen Eingang, welcher mit der GPI-Matrix **3016** über eine Nachrichten-Eingabeleitung **3314** verbunden ist, und einen Ausgang, der mit der GPI-Matrix **3016** über eine Nachrichtenausgabeleitung **3316** verbunden ist. Die gemeinsame kundenspezifische Interface- und Steuereinheit **3362** hat vorzugsweise einen Zeitsteuereingang, der mit einem Zeitsteuerausgang des dritten lokalen Zeitbasiseinheit **3360** über eine dritte Zeitsteuersignalleitung **3370** verbunden ist, einen ersten zweigerichteten Dateneingang, der mit einem zweigerichteten Dateneingang der Kopplungs-Ein/Ausgabe-Einheit **3304** verbunden ist, und einen Satz Verbindungen zur Ein/Ausgabe-Vorrichtung **3020**. In der bevorzugten Ausführungsform enthält der Satz Verbindungen zu der Ein/Ausgabe-Vorrichtung **3020** einen zweiten zweigerichteten Dateneingang, der mit einem zweigerichteten Dateneingang der Ein/Ausgabe-Vorrichtung **3020** verbunden ist, einen Adressenausgang, der mit einem Adresseneingang der Ein/Ausgabe-Vorrichtung **3020** verbunden ist, und einen zweigerichteten Steuereingang, der mit einem zweigerichteten Steuereingang der Ein/Ausgabe-T-Vorrichtung **3020** verbunden ist. Der Fachmann erkennt ohne weiteres, daß die Verbindungen zu der Ein/Ausgabe-Vorrichtung **3020** von der Art der Ein/Ausgabe-Vorrichtung **3020** abhängen, mit welcher die gemeinsame kundenspezifische Interface- und Steuereinheit **3362** verbunden ist.

Die dritte lokale Zeitbasiseinheit **3360** erhält das Master-Zeitsteuersignal von der Master-Zeitsteuereinheit **3025** und erzeugt ein drittes lokales Zeitsteuersignal. Die dritte lokale Zeitbasiseinheit **3360** gibt das dritte lokale Zeitsteuersignal an die Interface- und Steuereinheit **3362** ab, wodurch eine Zeitreferenz für die Ein/Ausgabe-T-Einrichtung **3018** geschaffen ist, in welcher sie untergebracht ist. In der bevorzugten Ausführungsform ist das dritte lokale Zeitsteuersignal phasensynchronisiert mit dem Master-Zeitsteuersignal. Jede dritte lokale Zeitbasiseinheit **3360** der T-Einrichtung arbeitet vorzugsweise auf einer identischen Frequenz. In einer alternativen Ausführungsform könnten jedoch auch ein oder mehr dritte lokale Zeitbasiseinheiten **3360** auf verschiedenen Frequenzen arbeiten. Die dritte lokale Zeitbasiseinheit **3360** ist vorzugsweise mit Hilfe einer herkömmlichen phasenstarren Frequenzumwandlungsschaltung ausgeführt, die eine auf CLB basierende phasenstarre Detektionsschaltung enthält. Analog zu den ersten und zweiten lokalen Zeitbasiseinheiten **3030** könnte auch die dritte lokale Zeitbasiseinheit **3360** als ein Teil eines Taktverteilungsbaums in einer alternativen Ausführungsform ausgeführt sein.

Der Aufbau und die Funktionalität der Ein/Ausgabe-Einheit **3304** in der T-Einrichtung **3018** ist vorzugsweise identisch mit derjenigen, die vorstehend für die T-Einrichtung **3014** beschrieben worden ist. Die Ein/Ausgabe-Einheit **3304** in der T-Einrichtung **3018** ist eine eindeutige Verbindungsadresse analog derjenigen für jede Verbindungs-Ein/Ausgabe-Einheit **3304** in jeder vorgegebenen T-Einrichtung **3014** zugeteilt.

Die gemeinsame kundenspezifische Interface- und Steuereinheit **3362** leitet den Nachrichtentransfer zwischen der Ein-Ausgabevorrichtung **3020**, an welche sie gekoppelt ist, und der Verb indungs-Ein/Ausgabe-Einheit **3304**, wobei eine Nachricht einen Befehl und möglicherweise Daten enthält. Die Interface- und Steuereinheit **3362** erhält Daten und Befehle von der entsprechenden Ein/Ausgabe-Vorrichtung **3020**. Vorzugsweise enthält jeder Befehl, der von der Ein/Ausgabe-Vorrichtung **3020** empfangen worden ist, eine Sollkopplungsadresse und einen Befehlscode, der eine ganz bestimmte durchzuführende Operationsart spezifiziert. In der bevorzugten Ausführungsform enthalten die Operationsarten, die eindeutig durch Befehlscodes identifiziert sind, 1) Datenanforderung, 2) Datenübertragungsbestätigung und 3) einen Unterbrechungssignal-Transfer. Die Sollverbindungsadresse identifiziert eine Sollverbindungs-Ein/Ausgabe-Einheit **3304** in dem System **3010**, in welche Daten und Befehle übertragen sind. Vorzugsweise überträgt die gemeinsame Interface- und Steuereinheit **3062** jeden Befehl und irgendwelche diesbezüglichen Daten als einen Satz auf einem Datenpaket basierender Nachrichten in herkömmlicher Weise, wobei jede Nachricht eine Sollverbindungsadresse und den Befehlscode enthält.

Zusätzlich zum Empfangen von Daten und Befehlen von der entsprechenden Ein/Ausgabe-Vorrichtung **3020** erhält die gemeinsame kundenspezifische Interface- und Steuereinheit **3362** Nachrichten von einer zugeordneten Verbindungs-Ein/Ausgabe-Einheit **3304**. In der bevorzugten Ausführungsform wandelt die Interface- und Steuereinheit **3363** eine Gruppe von verwandten Nachrichten in eine einzige Befehl- und Datenfolge entsprechend den Kommunikationsprotokollen, die von der entsprechenden Ein/Ausgabe-Vorrichtung **3020** getragen sind. In der bevorzugten Ausführungsform weist die gemeinsame kundenspezifische Interface- und Steuereinheit **3362** eine Steuereinheit für die auf CLB-basierende Ein/Ausgabe-Einheit auf, die mit der auf CLB-basierenden Schaltung verbunden ist, um Operationen durchzuführen, welche analog denjenigen sind, die von einer herkömmlichen SCI-Schalteinheit durchgeführt worden sind, wie durch ANSI/IEEE-Norm 1596–1992 definiert ist.

Die GPI-Matrix **3016** ist ein herkömmliches Verbindungsnetz, welches eine parallele Punkt-zu-Punkt-Nachrichtenlenkung zwischen Verbindungs-Ein/Ausgabe-Einheiten **3304** erleichtert. In der bevorzugten Ausführungsform ist die GPI-Matrix **3016** ein auf einer Verdrahtung basierendes, statisches k-näres (k-ary-) Würfel-Verbindungsnetz. In **Fig.** 24 ist ein Blockdiagramm einer beispielhaften Ausführungsform einer universellen Verbindungs-GPI-Matrix **3016** dargestellt. In **Fig.** 24 ist die Verbindungs-GPI-Matrix **3016** ein toroid- bzw. ringförmiges Verbindungsmachennetz, über dem entsprechend ein k-närer 2 Würfel, der eine Anzahl erster Übertragungskanäle **3380** und eine Anzahl zweiter Übertragungskanäle **3382** aufweist. Jeder erste Übertragungskanal **3380** enthält eine Anzahl Knotenverbindungsstellen **3384**, ebenso jeder zweite Übertragungskanal **3382**. Jede Verbindungs-Ein/Ausgabeeinheit **3304** in dem System **3010** ist vorzugsweise mit der Verbindungs-GPI-Matrix **3014** verbunden, so daß die Nachrichten-Eingangsleitung **3314** und die Nachrichten-Ausgangsleitung **3310** aufeinanderfolgende Knotenverbindungsstellen **3384** in einem vorgegebenen Übertragungskanal **3380, 3382** verbinden. In der bevorzugten Ausführungsform enthält jede T-Einrichtung **3014** eine Verbindungs-Ein/Ausgabe-Einheit **3304**, die mit dem ersten Verbindungskanal **3380** verbunden ist, und eine Verbindungs-Ein-/Ausgabeeinheit **3304**, die mit dem zweiten Verbindungskanal **3382** in der vorstehend beschriebenen Weise verbunden ist. Die gemein-

same Interface- und Steuereinheit **3302** in der T-Einrichtung **3014** erleichtert vorzugsweise das Leiten von Information zwischen der mit dem ersten Verbindungskanal verbundenen Ein/Ausgabe-Einheit **3304** und der mit dem zweiten Verbindungskanal **3382** verbundenen Ein/Ausgabeeinheit **3304**. Für eine T-Einrichtung **3014**, die eine Verbindungs-Ein/Ausgabe-Einheit **3304** hat, die mit dem ersten als **3380c** bezeichneten Übertragungskanal verbunden ist, und eine Verbindungs-Ein/Ausgabe-Einheit **3304** hat, die mit dem zweiten als **3382c** bezeichneten Übertragungskanal in **Fig.** 24 verbunden ist, erleichtert diese gemeinsame Interface- und Steuereinheit **3302** der T-Einrichtung, ein Informationslenken zwischen dieser Gruppe von ersten und zweiten Übertragungskanälen **3380c**, **3382c**.

Die Verbindungs-GPI-Matrix **3016** erleichtert folglich das Leiten von vielfachen Nachrichten in paralleler Form zwischen Verbindungs-Ein/Ausgabeeinheiten **3304**. Für die zweidimensionale GPI-Matrix **3016**, die in **Fig.** 24 dargestellt ist, enthält jede T-Einrichtung **3014** vorzugsweise eine einzige Verbindungs-Ein/Ausgabe-Einheit **3304** für den ersten Übertragungskanal **3380** und eine einzige Verbindungs-Ein/Ausgabe-Einheit **3304** für den zweiten Verbindungskanal **3382**. Der Fachmann erkennt, daß nur eine Ausführungsform, in welcher die Verbindungs-GPI-Matrix **3016** eine Anzahl Dimensionen hat, die größer als zwei ist, die T-Einrichtung **3014** vorzugsweise mehr als zwei Verbindungs-Ein/Ausgabe-Einheiten **3304** enthält. Vorzugsweise ist die Verbindungs-GPI-Matrix **3016** als eine k-närer 2-Würfel (k-ary 2-cube) mit einer 16 Bitdatenweggröße ausgeführt.

In der vorhergehenden Beschreibung sind verschiedene Elemente der vorliegenden Erfindung vorzugsweise unter Verwendung von rekonfigurierbaren Hardware-Ressourcen ausgeführt. Die Hersteller von umprogrammierbaren Logikvorrichtungen stellen üblicherweise veröffentliche Richtlinien zur Verfügung, um herkömmliche digitale Hardware mit Hilfe von umprogrammierbaren oder rekonfigurierbaren Hardware-Ressourcen auszuführen. Beispielsweise enthält das Xilinx-Buch "Programmierbare Logik-Daten" von 1994 (Xilinx, Inc., San Jose, CA) Anwendungshinweise, wie die folgenden: Anwendungshinweis XAPP 005.002 "Register-Based FIFO"; Anwendungshinweis XAPP 044.00 "High-Performance RAM-Based FIFO2"; Anwendungshinweis XAPP 013.001, "Using the Dedicated Carry Logic in the XC4000"; Anwendungshinweis XAPP 018.000 "Estimating the Performance of XC4000 Adders and Counters", Anwendungshinweis: XAPP 028.001, "Frequency/Phase Comparator for Phase-Locked Loops"; Anwendungshinweis: XAPP3 1.000 "Using the XC4000 RAM Capability", Anwendungshinweis: XAPP 036.001 "Four-Port DRAM Controller . . .", und Anwendungshinweis XAPP 039.001 "18-Bit Pipelinded Accumulator". Zusätzliches Material, das von Xilinx veröffentlicht worden ist, enthält Merkmale in "XCELL, The Quaterly Journal for Xilinx Programmable Logic Users". Beispielsweise erscheint ein Artikel, der sich im einzelnen mit der Ausführung schneller ganzzahliger Multiplikatoren befaßt, in Ausgabe 14, der Ausgabe vom 3. Quartal 1994.

Das hier beschriebene System **3010** ist eine skalierbare, parallele Computerarchitektur, um dynamisch Mehrfach-ISAs auszuführen. Eine einzelne S-Einrichtung **3012** kann ein ganzes Computerprogramm selbständig ablaufen lassen, unabhängig von einer anderen S-Einrichtung **3012** oder von externen Hardware-Ressourcen, wie ein Host-Computer. Auf einer einzelnen S-Einrichtung **3012** werden mehrere ISAs zeitlich nacheinander während einer Programmdurchführung entsprechend Rekonfigurationsunterbrechungen und/oder ins Programm eingebetteten Rekonfigurationsanweisungen ausgeführt. Da das System **3010** vorzugsweise mehrere S-Einrichtungen **3012** enthält, werden vorzugsweise mehrere Programme gleichzeitig durchgeführt, wobei jedes Programm unabhängig sein kann. Folglich werden, da das System **3010** vorzugsweise mehrere S-Einrichtungen **3012** enthält, mehrere IS-Architekturen immer gleichzeitig (d. h. parallel) außer während der System-Initialisierung oder Rekonfiguration durchgeführt. Das heißt, in einer vorgegebenen Zeit werden mehrere Sätze von Programmbefehlen gleichzeitig durchgeführt, wobei jeder Satz von Programmbefehlen gemäß einer entsprechenden IS-Architektur durchgeführt wird. Jede derartige IS-Architektur kann eindeutig sein.

S-Einrichtungen **3012** kommunizieren miteinander und mit Ein/Ausgabevorrichtungen **3020** über die Gruppe von T-Einrichtungen **3014**, die Verbindungs-GPI-Matrix **3016** und jede Ein/Ausgabe-T-Einrichtung **3018**. Obwohl jede S-Einrichtung **3012** ein ganzer Computer in sich ist, der in der Lage ist, einen unabhängigen Betrieb durchzuführen, kann eine S-Einrichtung **3012** als eine Master-S-Einrichtung **3012** für andere S-Einrichtungen **3012** oder das ganze System fungieren, das Daten und/oder Befehle zu anderen S-Einrichtungen **3012**, zu einer oder mehreren T-Einrichtungen **3016**, einer oder mehreren Ein/Ausgabe-T-Einrichtungen **3018** und ein oder mehreren Ein/Ausgabe-Einrichtungen **3022** sendet.

Das System **3010** gemäß der Erfindung ist folglich insbesondere bei Problemen verwendbar, die sowohl räumlich als zeitlich in ein oder mehrere daten-parallele Unterprobleme aufgeteilt werden können, beispielsweise in eine Bildverarbeitung, eine medizinische Datenverarbeitung, eine kalibrierte Farbanpassung, eine Datenbasis-Berechnung, eine Dokumenten-Verarbeitung, assoziative Sucheinrichtungen und Netzwerk-Server. Für Rechenprobleme mit einem großen Array von Operanden ist eine Daten-Parallelität vorhanden, wenn Algorithmen angewendet werden können, um so eine effektive Rechenbeschleunigung durch parallele Rechentechniken anzubieten. Probleme bei der parallelen Datenverarbeitung besitzen eine bekannte Komplexität, nämlich $O(n^k)$. Der Wert von k ist problem-abhängig, beispielsweise $k = 2$ für eine Bildverarbeitung und $k = 3$ für medizinische Datenverarbeitung. Bei der vorliegenden Erfindung werden die einzelnen S-Einrichtungen bzw. Geräte **3012** vorzugsweise verwendet, um eine Datenparallelität auf dem Niveau von Programmbefehlsgruppen auszunutzen. Da das System **3010** mehrere S-Einrichtungen **3012** enthält, wird das System **3010** vorzugsweise benutzt, um eine Datenparallelität auf dem Niveau von ganzen Programmsätzen auszunutzen.

Das System **3010** gemäß der Erfindung sorgt für sehr viel Rechenleistung, da es in der Lage ist, die Befehls-Verarbeitungshardware in jede S-Einrichtung **3012** vollständig zu rekonfigurieren, um die Rechenleistung solcher Hardware relativ zu Rechenerfordernissen zu einem vorgegebenen Zeitpunkt zu optimieren. Jede S-Einrichtung **3012** kann unabhängig von einer anderen S-Einrichtung **3012** rekonfiguriert werden. Das System **3010** behandelt in vorteilhafter Weise jeden Konfigurations-Datensatz, und folglich jede IS-Architektur, wie ein programmiertes Interface zwischen Software und der hier beschriebenen rekonfigurierbaren Hardware. Die Architektur der vorliegenden Erfindung erleichtert zusätzlich das Strukturieren von rekonfigurierbarer Hardware auf hohem Niveau, um selektiv die Angelegenheiten aktueller Systeme an Ort und Stelle zu adressieren, einschließlich Verhalten, bei welchen eine Unterbrechung eine Befehlsverarbeitung beeinflußt, die Notwendigkeit für eine deterministische Latenzantwort, um eine Echtzeit-Verarbeitung und Steuerfähigkeit zu erleichtern, und die Notwendigkeit für wählbare Antworten bei einer Fehler-Behandlung.

Im Unterschied zu anderen Computer-Architekturen lehrt die vorliegende Erfindung jederzeit die maximale Ausnut-

zung von Silizium-Ressourcen. Die vorliegende Erfindung sorgt für ein paralleles Computersystem, das jederzeit auf jede gewünschte Größe vergrößert werden kann, sogar auf Tausende von S-Einrichtungen **12**. Eine derartige architektonische Skalierbarkeit ist möglich, da eine auf S-Geräten basierende Befehlsverarbeitung absichtlich von einer auf T-Einrichtungen basierenden Datenkommunikation getrennt ist. Dieses Paradigma-Befehlsverarbeitung Datenkommunikation-Trennung ist äußerst günstig für eine parallele Datenberechnung. Der interne Aufbau einer S-Geräte-Hardware ist vorzugsweise für den zeitlichen Durchlauf von. Befehlen optimiert, während der interne Aufbau einer T-Geräte-Hardware vorzugsweise für eine effiziente Datenkommunikation optimiert ist. Die Menge an S-Einrichtungen bzw. -Geräten **3012** und die Menge an T-Einrichtungen bzw. Geräten sind jeweils eine trennbare, konfigurierbare Komponente in einer Raum-Zeitaufteilung einer daten-parallelen Rechenarbeit.

Mit Hilfe der vorliegenden Erfindung kann zukünftige rekonfigurierbare Hardware ausgenutzt werden, um Systeme mit noch größeren Rechenkapazitäten zu bauen, wobei der hier beschriebene Gesamtaufbau erhalten bleibt. Mit anderen Worten, das System **10** gemäß der Erfindung ist technologisch skalierbar. Praktisch brauchen alle gegenwärtig verfügbaren, rekonfigurierbaren Logik-Vorrichtungen auf speicher-basierender CMOS-(Komplementärer Metalloxid-Halbleiter)Technologie. In künftigen Systemen würde eine rekonfigurierbare logische Vorrichtung, die verwendet ist, um eine S-Einrichtung auszuführen, eine Aufteilung von internen Hardware-Ressourcen gemäß den hier beschriebenen Innenschleifen- und Außenschleifen-ISA-Parametriken haben. Größere rekonfigurierbare logische Vorrichtungen bieten einfach die Möglichkeit, mehr daten-parallele Rechenarbeit in einer einzigen Vorrichtung durchzuführen. Beispielsweise würde eine größere funktionelle Einheit **3194** in der zweiten beispielhaften Ausführungsform der DO-Einheit **3063**, wie vorstehend anhand von **Fig. 17B** beschrieben ist, größere Abbildungs-Kerngrößen unterbringen. Der Fachmann erkennt, daß die technologische Skalierbarkeit, welche durch die vorliegende Erfindung geschaffen ist, weder auf auf CMOS-basierende Vorrichtungen noch auf FPGA basierende Ausführungen beschränkt ist. Folglich schafft die Erfindung eine technologische Skalierbarkeit ungeachtet der speziellen Technologie, die verwendet ist, um eine Rekonfigurierbarkeit oder Umprogrammierbarkeit zu schaffen.

In **Fig.** 25A und 25B ist ein Ablaufdiagramm eines bevorzugten Verfahrens für ein skalierbares, paralleles, dynamisch rekonfigurierbares Berechnen dargestellt. Vorzugsweise wird das in **Fig.** 25A und 25B dargestellte Verfahren in jeder der S-Einrichtungen **3012** in dem System **3010** durchgeführt. Das bevorzugte Verfahren beginnt beim Schritt **4000** in **Fig.** 25A mit der Rekonfigurationslogik **3104**, die einen Konfigurations-Datensatz wieder auffindet, der einer ISA entspricht. Als nächstes konfiguriert beim Schritt **4002** die Rekonfigurationslogik **3104** jedes Element in der IF-Einheit **3060** der DO-Einheit **3062** und der AO-Einheit **3064** entsprechend dem wiederaufgefundenen Konfigurations-Datensatz beim Schritt **4002**, wodurch eine DRPU-Hardware-Organisation für die Ausführung der aktuellen in Betracht gezogenen ISA erzeugt wird. Im Anschluß an den Schritt **4002** findet die Unterbrechungslogik **3106** die Unterbrechungsantwortsignale wieder, die in dem Architektur-Beschreibungsspeicher **3101** gespeichert sind, und erzeugt einen entsprechenden Satz von Übergangssteuersignalen, welche festlegen, wie die aktuelle DRPU-Konfiguration auf Unterbrechungen beim Schritt **4004** anspricht. Das ISS **3100** initialisiert anschließend beim Schritt **4006** eine Programm-Zustandsinformation, worauf das ISS **3100** einen Befehlsausführungszyklus beim Schritt **4008** initialisiert.

Als nächstes bestimmt das ISS **3100** oder die Unterbrechungslogik **3106**, ob Rekonfiguration erforderlich ist. Das ISS **3100** legt fest, daß Rekonfiguration in dem Fall erforderlich ist, daß eine Rekonfigurationsanweisung während einer Programmausführung gewählt wird. Die Unterbrechungslogik **3106** bestimmt, daß Rekonfiguration entsprechend einer Rekonfigurations-Unterbrechung erforderlich ist. Wenn Rekonfiguration erforderlich ist, geht das bevorzugte Verfahren auf Schritt **4012** über, bei welchem ein Rekonfigurations-Handhabungsprogramm Programmzustandsinformation rettet. Vorzugsweise enthält die Programmzustandsinformation eine Referenz zu dem Konfigurationsdatensatz, welcher der aktuellen DRPU-Konfiguration entspricht. Nach dem Schritt **4012** kehrt das bevorzugte Verfahren auf den Schritt **4000** zurück, um einen nächsten Konfigurations-Datensatz aufzufinden, auf den durch die Rekonfigurations-Anweisung oder die Rekonfigurations-Unterbrechung Bezug genommen ist.

Für den Fall, daß die Rekonfiguration beim Schritt **4010** nicht erforderlich ist, bestimmt die Unterbrechungslogik **3106**, ob eine Nicht-Rekonfigurations-Unterbrechung eine Wartung beim Schritt **4014** erfordert. Wenn dem so ist, bestimmt das ISS **3100** als nächstes beim Schritt **4020**, ob ein Zustandsübergang von dem aktuellen ISS-Zustand in dem Befehlsausführungszyklus in den Unterbrechungs-Servicezustand basierend auf den Übergangssteuersignalen zulässig ist. Wenn ein Zustandsübergang in den Unterbrechungs-Servicezustand nicht zulässig ist, geht das ISS **3100** in den nächsten Zustand in dem Befehlsausführungszyklus über und kehrt auf den Schritt **4020** zurück. In dem Fall, daß Übergangssteuersignale einen Zustandsübergang von dem aktuellen ISS-Zustand in dem Befehlsausführungszyklus in den Unterbrechungs-Servicezustand erlauben, geht ISS **3100** als nächstes beim Schritt **4024** in den Unterbrechungs-Servicezustand über. Beim Schritt **4024** stellt das ISS **3100** eine Programmzustandsinformation sicher und führt Programmbefehle für ein Warten der Unterbrechung aus. Im Anschluß an den Schritt **4024** kehrt das bevorzugte Verfahren auf den Schritt **4008** zurück, um den aktuellen Befehlsausführungszyklus wieder aufzunehmen, wenn er noch nicht beendet worden ist, oder um einen nächsten Befehlsausführungszyklus zu initiieren.

Für den Fall, daß keine Nicht-Konfigurations-Unterbrechung eine Wartung beim Schritt **4014** erfordert, geht das bevorzugte Verfahren auf den Schritt **4016** über und stellt fest, ob die Durchführung des aktuellen Programms beendet ist. Wenn die Durchführung des aktuellen Programms noch andauert, kehrt das bevorzugte Verfahren auf den Schritt **4008** zurück, um einen anderen Befehlsausführungszyklus zu initiieren. Andererseits ist das bevorzugte Verfahren beendet.

Die Lehren der vorliegenden Erfindung unterscheiden sich deutlich von anderen Systemen und Verfahren für ein umprogrammierbares oder rekonfigurierbares Rechnen. Insbesondere ist die vorliegende Erfindung nicht äquivalent mit einer herunterladbaren Mikrocode-Architektur, da solche Architekturen im allgemeinen auf nicht-rekonfigurierbare Steuereinrichtung und eine nicht-rekonfigurierbare Hardware angewiesen sind. Die vorliegende Erfindung unterscheidet sich also deutlich von einem angeschlossen rekonfigurierbaren Prozessor-(ARP-)System, in welchem eine Gruppe von rekonfigurierbaren Hardware-Ressourcen mit einem nicht-rekonfigurierbaren Host-Prozessor oder Host-System verbunden ist. Die ARP-Einrichtung hängt von dem Host ab, um gewisse Programmbefehle durchzuführen. Daher wird eine Menge verfügbarer Silizium-Ressourcen nicht maximal über den Zeitrahmen der Programmdurchführung genutzt, da Silizium-

Ressourcen bei der ARP-Einrichtung oder dem Host unbenutzt sind oder ineffizient genutzt werden, wenn der Host bzw. die ARP-Einrichtung mit Daten arbeitet. Im Unterschied hierzu ist jede S-Einrichtung **3012** ein unabhängiger Rechner, in welchem ganze Programme ohne weiteres ausgeführt werden können. Mehrere S-Einrichtungen **3012** führen vorzugsweise gleichzeitig Programme durch. Die vorliegende Erfindung lehrt daher die ständige maximale Ausnutzung von Silizium-Ressourcen sowohl für einzelne Programme, die von einzelnen S-Einrichtungen **3012** durchgeführt werden oder von mehreren Programmen, die von dem gesamten System **3010** ausgeführt werden.

Eine ARP-Einrichtung stellt einen Rechenbeschleuniger für einen ganz bestimmten Algorithmus in einer ganz bestimmten Zeit zur Verfügung und ist als ein Satz von Verknüpfungsgliedern ausgeführt, die optimal bezüglich dieses spezifischen Algorithmus miteinander verbunden sind. Die Verwendung von rekonfigurierbaren Hardware-Ressourcen für universelle Operationen, wie eine verwaltende Befehlsausführung, ist bei ARP-System vermieden. Darüber hinaus behandelt ein ARP-System nicht eine vorgegebene Menge von miteinander verbundenen Verknüpfungsgliedern bzw. Gates als eine ohne weiteres wiederverwendbare Ressource. Im Gegensatz, die vorliegende Erfindung lehrt eine dynamisch rekonfigurierbare Verarbeitungseinrichtung, die für ein effizientes Management einer Befehlsausführung gemäß einem Befehlsausführungsmodells konfiguriert ist, das am besten für die Rechenerfordernisse zu einem ganz bestimmten Zeitpunkt ausgelegt ist. Jede S-Einrichtung **3012** weist eine Vielzahl ohne weiteres wiederverwendbarer Ressourcen, beispielsweise das ISS **3100**, die Unterbrechungslogik **3106** und die Speicher/Ausrichtlogik **3152** auf. Die vorliegende Erfindung lehrt die Verwendung von rekonfigurierbaren logischen Ressourcen auf dem Niveau von LCBs- oder IOBs-Gruppen und rekonfigurierbarer Verbindungen, jedoch nicht auf dem Niveau von miteinander verbundenen Gates. Die vorliegende Erfindung lehrt folglich die Verwendung von rekonfigurierbaren höherwertigen logischen Designkonstrakts, die zum Durchführen von Operationen der ganzen Klassen von Rechenproblemen verwendbar sind, und lehrt nicht ein brauchbares Verbindungsschema, das für einen einzigen Algorithmus verwendbar ist.

Im allgemeinen sind ARP-Systeme auf ein Übertragen eines ganz bestimmten Algorithmus in einen Satz von miteinander verbundenen Gates gerichtet. Einige ARP-Systeme versuchen, hochwertige Befehle in einer optimalen Hardware-Konfiguration zu compilieren, welches im allgemeinen ein hartes NP-Problem ist. Im Unterschied hierzu lehrt die Erfindung die Verwendung eines Compilers für ein dynamisch rekonfigurierbares Berechnen, das hochwertige Programmbefehle in Assembler-Sprachenbefehle gemäß einer variablen ISA auf sehr unkomplizierte Weise compiliert.

Eine ARP-Einrichtung ist im allgemeinen nicht in der Lage, ihre eigenes Host-Programm als Daten zu behandeln oder es selbst zu kontextualisieren. Im Unterschied hierzu kann jede S-Einrichtung in dem System **3010** ihre eigenen Programme als Daten behandeln, und folglich ohne weiteres selbst kontextualisieren. Das System **3010** kann ohne weiteres sich selbst durch die Ausführung seiner eigenen Programme simulieren. Die vorliegende Erfindung hat zusätzlich die Fähigkeit, ihren eigenen Compiler zu compilieren.

In der vorliegenden Erfindung kann ein einziges Programm eine erste Gruppe von Befehlen, die zu einem ersten ISA gehören, eine zweite Gruppe von Befehlen, die zu einer zweiten ISA gehören, eine dritte Gruppe von Befehlen, die zu noch einer weiteren ISA gehören, usw. enthalten. Die hier beschriebene Architektur führt jede derartige Gruppe von Befehlen mit Hilfe von Hardware durch, die hinsichtlich Durchlaufzeit konfiguriert ist, um die ISA durchzuführen, zu welcher die Befehle gehören. Keine bekannten Systeme oder Methoden bieten ähnliche Lehren an.

Die Erfindung lehrt ferner ein rekonfigurierbares Unterbrechungsschema, bei welchem Unterbrechungslatenz, Unterbrechungspräzision und ein programmierbares Zustandsübergangs-Freigeben gemäß der aktuellen, in Betracht gezogenen ISA sich ändern kann. Keine anlogen Lehren werden in anderen Computersystemen gefunden. Die vorliegende Erfindung lehrt zusätzlich ein Computersystem mit einer rekonfigurierbaren Datenweg-Bitbreite, einer Adressen-Bitbreite und rekonfigurierbare Steuerzeilen-Breiten im Unterschied zu herkömmlichen Computersystemen.

Zusammenfassend wurde ein Kompiliersystem und ein Verfahren zur Erzeugung einer Folge von Programmbefehlen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit geschaffen, die eine interne Hardwareorganisation aufweist, die wahlweise unter einer Anzahl von Hardwarearchitekturen geändert werden kann, wobei jede Hardwarearchitektur Befehle von einem entsprechenden Befehlssatz ausführt. Quelldateien werden zur Ausführung mit Hilfe von mehreren Befehlsatzarchitekturen (instruction set architectures) kompiliert, wie dies durch Rekonfigurations-Übersetzungsanweisungen spezifiziert wird. Die Objektdateien fassen wahlweise Bitströme, die Hardwarearchitekturen spezifizieren, die Befehlssatzarchitekturen entsprechen, mit ausführbarem Code zur Ausführung auf den Architekturen zusammen.

Bezugszeichenliste

Fig. 1
12 S-Einrichtung
14 T-Einrichtung
18 Ein/Ausgabe-T-Einrichtung
20 Ein/Ausgabeeinrichtung
16 Mehrzweckverbindungsmatrix (GPI-Matrix)
22 Master-Zeitbasiseinheit
Fig. 1A
131 Taktgenerator
132 Bitstromspeicher
133 Programm-/Datenspeicher
12 S-Einrichtung
14 T-Einrichtung
Fig. 1B
149 Speicherbus
140 FPGA Konfigurationshardware

146 Befehle dekodieren
147 Speicherinterface
132 ISA0 Bitstrom;
ISA1 Bitstrom
Fig. 1C
siehe **Fig.** 1B
Fig. 3
301 Quelldatei lesen
302 ISA identifizieren
303 Rekonfigurationsanweisung erzeugen
304 Anweisungen für identifizierte ISA kompilieren
305 weitere ISA?
306 übersetzen
307 binden
308 laden
309 Ende
Fig. 3A
600 nächste Hochsprachenanweisung auswählen
601 Funktionsaufruf
602 verschiedene ISA?
603 RTL-Code abgeben
605 RTL-Funktionsaufruf abgeben
607 RTL-Code zum Speichern lebender Register abgeben
604 RTL-Rekonfigurationscode abgeben
606 RTL-Funktionsaufruf abgeben
608 weitere Hochsprachenanweisung?
609 RTL-Rekonfigurationscode abgeben
611 RTL-Code zum Wiederherstellen lebender Register abgeben
613 RTL-Code zum Lesen von Rückgabewert abgeben
Fig. 3B
612 nächste RTL-Anweisung auswählen
618 Regel für aktuelle RTL-Anweisungsgruppe erhalten
620 Maschinensprachenanweisung erzeugen. Gemäß der Regel für diese ISA setzen
622 andere RTL-Anweisung?
610 Registerreservierung ausführen
Fig. 3C
331 RTL-Code mit neuer ISA mit Bemerkungen versehen
332 ISA-abhängige und ISA-unabhängige Optimierung
333 maschinenabhängige Anweisungen erzeugen
Fig. 4
401 Quelle
403 Objekt
404 Binder
405 ausführbares Programm
406 ISA-Bitströme
10 rekonfigurierbarer Computer
407 Ladeprogramm
Fig. 5
501 Binden-Ansicht
503 ELF-Kopfteil
504 Programmkopfteiltabelle optional
505 Abschnitt **1**
. . .
Abschnitt n
. . .
506 Abschnittskopfteiltabelle
502 Ausführen-Ansicht
503 ELF-Kopfteil
504 Programmkopfteiltabelle
505 Abschnitt **1**
Abschnitt **2**
. . .
506 Abschnittskopfteiltabelle optional
Fig. 6
651 Anweisung reconfig
654 vorbestimmte Adresse <= SP
655 Hardware lädt nächste Konfiguration
656 SP <= vorbestimmte Adresse

Fig. 7
707 lebende Registerwerte retten
701 Parameter zu aufgerufener Funktion schieben
702 rekonfigurieren zu neuer ISA
703 Unterprogramm aufrufen
704 Fluß zum Aufrufer zurückkehren
705 auf ursprüngliche ISA rekonfigurieren
706 Rückgabewert lesen
708 lebende Registerwerte wieder herstellen

Patentansprüche

1. Kompilierverfahren zur Erzeugung einer Folge von Programmbefehlen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit (**10**), die eine interne Hardwareorganisation aufweist, die während der Ausführung der Folge von Programmbefehlen wahlweise unter einer Anzahl von Hardwarearchitekturen geändert werden kann, wobei jede Hardwarearchitektur Befehle aus einem entsprechenden Befehlssatz ausführt, mit den folgenden Schritten:
   a) als Eingabe wird eine Quelldatei (**301**) empfangen, die eine Anzahl von Quellcode-Befehlsanweisungen enthält, und zwar einschließlich mindestens eines ersten Untersatzes von Befehlsanweisungen und eines zweiten Untersatzes von Befehlsanweisungen;
   b) für den ersten Untersatz von Befehlsanweisungen wird ein erster Befehlssatz identifiziert;
   c) für den zweiten Untersatz von Befehlsanweisungen wird ein zweiter Befehlssatz identifiziert; und
   d) der erste Untersatz von Befehlsanweisungen zur Ausführung wird mit Hilfe des ersten Befehlssatzes kompiliert bzw. übersetzt und der zweite Untersatz von Befehlsanweisungen zur Ausführung wird mit Hilfe des zweiten Befehlssatzes kompiliert.
2. Verfahren nach Anspruch 1, für das
beim Schritt b) von dem Quellcode eine Rekonfigurations-Betriebsanweisung abgerufen wird, die den ersten Befehlssatz spezifiziert; und
beim Schritt c) von dem Quellcode eine Rekonfigurations-Betriebsanweisung abgerufen wird, die den zweiten Befehlssatz spezifiziert.
3. Verfahren nach Anspruch 2, bei dem jede Rekonfigurations-Betriebsanweisung unter Verwendung einer Meta-Syntax bereitgestellt wird.
4. Verfahren nach Anspruch 2 oder 3, bei dem jede Rekonfigurations-Betriebsanweisung entweder eine Direkt-Rekonfigurations-Betriebsanweisung, eine Funktionsebenen-Rekonfigurations-Betriebsanweisung oder eine Standard-Rekonfigurations-Betriebsanweisung umfaßt.
5. Verfahren nach einem der vorhergehenden Ansprüche, mit dem weiteren Schritt:
   e) es wird eine ausführbare Datei erzeugt, die die Ergebnisse des Schrittes d) beinhaltet und außerdem für jeden Untersatz von Befehlsanweisungen einen Rekonfigurationscode, der den Befehlssatz identifiziert, der dem Untersatz von Befehlsanweisungen entspricht.
6. Verfahren nach einem der Ansprüche 1 bis 4, mit dem weiteren Schritt:
   e) es wird eine ausführbare Datei erzeugt, die die Ergebnisse des Schrittes d) beinhaltet und außerdem für jeden Untersatz von Befehlsanweisungen einen Verweis, der einen Bitstrom bestimmt, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.
7. Verfahren nach einem der Ansprüche 1 bis 4, mit dem weiteren Schritt:
   e) es wird eine ausführbare Datei erzeugt, die die Ergebnisse des Schrittes d) beinhaltet und außerdem für jeden Untersatz von Befehlsanweisungen einen Verweis, der entsprechend einem erweiterten, ausführbaren Programm und einem Bindungsformat codiert ist, wobei der Verweis einen Bitstrom bestimmt, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.
8. Verfahren nach einem der Ansprüche 1 bis 4, mit dem weiteren Schritt:
   e) es wird eine ausführbare Datei erzeugt, die die Ergebnisse des Schrittes d) beinhaltet und außerdem für jeden Untersatz von Befehlsanweisungen einen Bitstrom, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.
9. Verfahren nach einem der Ansprüche 1 bis 4, mit den weiteren Schritten:
   e) eine erste Objektdatei wird erzeugt, die die Ergebnisse des Schrittes d) beinhaltet und außerdem für jeden Untersatz von Befehlsanweisungen einen Rekonfigurationscode, der den Befehlssatz identifiziert, der dem Untersatz von Befehlsanweisungen entspricht;
   f) die Schritte a) bis e) werden mindestens für eine zweite Quelldatei wiederholt, um mindestens eine zweite Objektdatei zu erzeugen; und
   g) die in den Schritten e) und f) erzeugten Objektdateien werden gebunden, um eine ausführbare Datei zu erzeugen.
10. Verfahren nach Anspruch 9, mit dem weiteren Schritt:
   h) an der erzeugten, ausführbaren Datei wird entsprechend den Ausrichtungserfordernissen eine Speicherausrichtung vorgenommen.
11. Verfahren nach Anspruch 10, bei dem die erzeugte, ausführbare Datei einem Bitstrom zugeordnet ist, der einen Befehlssatz darstellt, und bei dem der Schritt h) den Schritt umfaßt:
   h.1) der Bitstrom wird aufgefüllt, um eine Speicherausrichtung durchzuführen.
12. Verfahren nach Anspruch 9, bei dem:
beim Schritt a) als Eingabe eine Quelldatei empfangen wird, die eine Anzahl von Quellcode-Befehlsanweisungen

einschließlich mindestens eines ersten Untersatzes von Befehlsanweisungen und eines zweiten Untersatzes von Befehlsanweisungen enthält, wobei zumindest eine der Befehlsanweisungen einen externen Verweis enthält; und
beim Schritt e) eine erste Objektdatei erzeugt wird, die die Ergebnisse des Schrittes d) enthält und außerdem für jeden Untersatz von Befehlsanweisungen einen Rekonfigurationscode, der den Befehlssatz identifiziert, der dem Untersatz von Befehlsanweisungen entspricht, wobei zumindest eine der Befehlsanweisungen einen externen Verweis enthält; und
mit dem weiteren Schritt:

f.1) vor der Ausführung des Schrittes g) werden für jede Objektdatei die externen Verweise aufgelöst.

13. Verfahren nach einem der vorhergehenden Ansprüche, bei dem der erste Untersatz von Befehlsanweisungen eine erste definierte Funktion und der zweite Untersatz von Befehlsanweisungen eine zweite definierte Funktion umfaßt.

14. Verfahren nach einem der Ansprüche 1 bis 13, bei dem der erste Untersatz von Befehlsanweisungen einen ersten beliebigen Block von Anweisungen und der zweite Untersatz von Befehlsanweisungen einen zweiten beliebigen Block von Anweisungen umfaßt.

15. Verfahren nach Anspruch 14, bei dem die Quelldatei mindestens einen Funktionsaufruf und einen Funktionsrücksprung umfaßt und bei dem die Schritte b) und c) jeweils die wahlweise Ausführung einer interprozeduralen Analyse umfassen, um bei jedem Funktionsaufruf und bei jedem Funktionsrücksprung eine im Kontext befindliche Befehlsatzarchitektur (ISA) zu identifizieren.

16. Verfahren nach Anspruch 1, mit den weiteren Schritten:

e) der erste kompilierte Untersatz von Befehlsanweisungen wird für den ersten Befehlssatz optimiert; und

f) der zweite kompilierte Untersatz von Befehlsanweisungen wird für den zweiten Befehlssatz optimiert.

17. Kompilierverfahren zur Erzeugung einer Folge von Programmbefehlen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit, die eine interne Hardwareorganisation aufweist, die während der Ausführung der Folge von Programmbefehlen unter einer Anzahl von Hardwarearchitekturen verändert werden kann, wobei jede Hardwarearchitektur Befehle von einem entsprechenden Befehlssatz ausführt, mit den folgenden Schritten:

a) eine Quellcode-Befehlsanweisung wird von einer Quelldatei ausgewählt, die eine Anzahl von Quellcode-Befehlsanweisungen enthält;

b) in Antwort auf die Anweisung, die einen Funktionsaufruf umfaßt, werden die folgenden Schritte ausgeführt:

b.1) ein erster Befehlssatz, der augenblicklich im Kontext steht, wird bestimmt;

b.2) ein zweiter Befehlssatz für den Funktionsaufruf wird bestimmt;

b.3) in Antwort darauf, daß der erste Befehlssatz von dem zweiten Befehlssatz abweicht, werden die folgenden Schritte ausgeführt:

b.3.1) Codeanweisungen zur Rekonfiguration werden an den zweiten Befehlssatz abgegeben;

b.3.2) für den Funktionsaufruf wird eine kompilierte Codeanweisung abgegeben; und

b.3.3) zur Rekonfiguration werden an den ersten Befehlssatz Codeanweisungen abgegeben; und

b.4) in Antwort darauf, daß die erste Befehlsatzarchitektur identisch zu dem zweiten Befehlssatz ist, wird für den Funktionsaufruf eine kompilierte Codeanweisung abgegeben;

c) in Antwort darauf, daß die Anweisung keinen Funktionsaufruf umfaßt, wird für die Anweisung eine kompilierte Codeanweisung abgegeben; und

d) die Schritte a) bis c) werden für jede Quellcode-Befehlsanweisung in der Quelldatei wiederholt.

18. Verfahren nach Anspruch 17, bei dem der Schritt b.3.1) die folgenden Schritte umfaßt:
b.3.1.1.) es wird eine Codeanweisung zum Retten von lebenden Registern abgegeben; und
b.3.1.2) es wird eine Rekonfigurations-Codeanweisung abgegeben; und bei dem der Schritt b.3.3) die folgenden Schritte umfaßt:
b.3.3.1) es wird eine Rekonfigurations-Codeanweisung abgegeben; und
b.3.3.2) es wird eine Codeanweisung zur Wiederherstellung der geretteten, lebenden Register abgegeben.

19. Verfahren nach Anspruch 18, bei dem die Codeanweisungen Registertransferebenenanweisungen (RTL-Anweisungen) umfassen.

20. Verfahren nach Anspruch 19, mit den weiteren Schritten:

e) eine Register- bzw. Speicherreservierung wird durchgeführt;

f) für jede Registertransferebenenanweisung wird:

f.1) bestimmt, ob für die Registertransferebenenanweisung eine Übersetzungsregel existiert; und

f.2) in Antwort auf die Feststellung, daß eine Übersetzungsregel existiert, wird für die Registertransferebenenanweisung entsprechend der Übersetzungsregel ein Assemblercode erzeugt.

21. Verfahren nach Anspruch 19, mit den weiteren Schritten:

e) jede Registertransferebenenanweisung wird mit Bemerkungen versehen, um eine Befehlsatzarchitektur (ISA) anzugeben;

f) die Registertransferebenenanweisungen werden optimiert; und

g) aus den optimierten Registertransferebenenanweisungen wird ein maschinenabhängiger Assemblercode erzeugt.

22. Verfahren zur Rekonfigurierung einer dynamisch rekonfigurierbaren Verarbeitungseinheit während der Ausführung eines Programms, das eine Folge von Programmbefehlen umfaßt, mit den folgenden Schritten:

a) es wird eine Rekonfigurations-Betriebsanweisung identifiziert, die einen neuen Befehlssatz spezifiziert;

b) ein Zustand der Programmausführung wird gespeichert;

c) der neue Befehlssatz wird geladen;

d) der gespeicherte Zustand der Programmausführung wird wieder abgerufen; und

e) die Ausführung wird unter Verwendung des neuen Befehlssatzes wieder aufgenommen.

23. Verfahren nach Anspruch 22, bei dem der Schritt b) die folgenden Schritte umfaßt:
b.1) Zustandsvariablen werden auf einem Stapel gespeichert, auf den mit Hilfe eines Stapelzeigers (SP) verwiesen wird; und
b.2) der Stapelzeiger wird in einer Speicherstelle abgespeichert; und bei dem der Schritt d) die Schritte umfaßt:
d.1) der Stapelzeiger wird von der Speicherstelle wieder abgerufen; und
d.2) die Zustandsvariablen werden von dem Stapel wieder abgerufen.

24. Kompiliersystem zur Erzeugung einer Folge von Programmanweisungen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit, die eine interne Hardwareorganisation aufweist, die während der Ausführung der Folge von Programmanweisungen unter einer Anzahl von Hardwarearchitekturen geändert werden kann, wobei jede Hardwarearchitektur Befehle von einem entsprechenden Befehlssatz ausführt, welches Kompiliersystem umfaßt:
a) ein Eingabegerät zur Eingabe von mindestens einer Quelldatei, die eine Anzahl von Quellcode-Befehlsanweisungen enthält, und zwar einschließlich von mindestens einem ersten Untersatz von Befehlsanweisungen, einem zweiten Untersatz von Befehlsanweisungen, sowie für jeden Untersatz von Befehlsanweisungen eine Rekonfigurations-Betriebsanweisung, die eine der Hardwarearchitekturen spezifiziert; und
b) einen Kompiler, der verbunden ist, um von dem Eingabegerät jede Quelldatei zu empfangen, zur Kompilierung jeder Eingabe-Quelldatei, um eine Objektdatei zu erzeugen, indem der Befehlssatz, der der von jeder Rekonfigurations-Betriebsanweisung spezifizierten Hardwarearchitektur entspricht, identifiziert wird, indem mindestens ein Abschnitt der Eingabe-Quelldatei zur Ausführung unter Verwendung jedes identifizierten Befehlssatzes kompiliert wird, und indem ein Rekonfigurationscode erzeugt wird, der jeder Rekonfigurations-Betriebsanweisung entspricht.

25. Kompiliersystem nach Anspruch 24, das außerdem umfaßt:
einen Binder bzw. Linker, der verbunden ist, um jede Objektdatei zu empfangen, zur Verknüpfung der Objektdateien, um eine ausführbare Datei zu erzeugen, die für jede Rekonfigurations-Betriebsanweisung einen Verweis auf einen Bitstrom enthält, der die durch die Rekonfigurations-Betriebsanweisung spezifizierte Hardwarearchitektur beschreibt, sowie eine Folge von Programmanweisungen für die beschriebene Hardwarearchitektur.

26. Kompiliersystem nach Anspruch 24, das außerdem umfaßt:
einen Binder bzw. Linker, der verbunden ist, um jede Objektdatei zu empfangen, zur Verknüpfung der Objektdateien, um eine ausführbare Datei zu erzeugen, die für jede Rekonfigurations-Betriebsanweisung einen Bitstrom enthält, der die von der Rekonfigurations-Betriebsanweisung spezifizierte Hardwarearchitektur beschreibt, sowie eine Folge von Programmbefehlen für die beschriebene Hardwarearchitektur.

27. Computerprogrammprodukt, das ein computerverwendbares Medium umfaßt, auf dem ein computerlesbarer Code verkörpert ist, zur Erzeugung einer Folge von Programmbefehlen zur Verwendung in einer dynamisch rekonfigurierbaren Verarbeitungseinheit, welche umfaßt:
a) Geräte für computerlesbaren Programmcode, die ausgelegt sind, um als Eingabe eine Quelldatei zu empfangen, die eine Anzahl von Quellcode-Befehlsanweisungen enthält, einschließlich mindestens eines ersten Untersatzes von Befehlsanweisungen und eines zweiten Untersatzes von Befehlsanweisungen;
b) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um einen ersten Befehlssatz für den ersten Untersatz von Befehlsanweisungen zu identifizieren;
c) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um einen zweiten Befehlssatz für den zweiten Untersatz von Befehlsanweisungen zu identifizieren; und
d) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um den ersten Untersatz von Befehlsanweisungen zur Ausführung unter Verwendung des ersten Befehlssatzes zu kompilieren und um den zweiten Untersatz von Befehlsanweisungen zur Ausführung unter Verwendung des zweiten Befehlssatzes zu kompilieren.

28. Computerprogrammprodukt nach Anspruch 27, das außerdem umfaßt:
Geräte für computerlesbaren Programmcode, die konfiguriert sind, um eine ausführbare Datei zu erzeugen, welche die kompilierten Anweisungen enthält und außerdem für jeden Untersatz von Befehlsanweisungen einen Rekonfigurationscode, der den Befehlssatz identifiziert, der dem Untersatz von Befehlsanweisungen entspricht.

29. Computerprogrammprodukt nach Anspruch 27, das außerdem umfaßt:
Geräte für computerlesbaren Programmcode, die konfiguriert sind, um eine ausführbare Datei zu erzeugen, welche die kompilierten Anweisungen enthält und außerdem für jeden Untersatz von Befehlsanweisungen einen Verweis, der einen Bitstrom bezeichnet, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.

30. Computerprogrammprodukt nach Anspruch 27, das außerdem umfaßt:
Geräte für computerlesbaren Programmcode, die konfiguriert sind, um eine ausführbare Datei zu erzeugen, die die kompilierten Anweisungen enthält und außerdem für jeden Untersatz von Befehlsanweisungen einen Verweis, der entsprechend einem erweiterten ausführbaren Programm und einem Bindeformat codiert ist, wobei der Verweis einen Bitstrom bezeichnet, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.

31. Computerprogrammprodukt nach Anspruch 27, das außerdem umfaßt:
Geräte für computerlesbaren Programmcode, die konfiguriert sind, um eine ausführbare Datei zu erzeugen, die die kompilierten Anweisungen enthält und außerdem für jeden Untersatz von Befehlsanweisungen einen Bitstrom, der den Befehlssatz darstellt, der dem Untersatz von Befehlsanweisungen entspricht.

32. Computerprogrammprodukt nach Anspruch 27, das außerdem umfaßt:
a) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um eine erste Objektdatei zu erzeugen, welche die kompilierten Anweisungen enthält, und außerdem für jeden Untersatz von Befehlsanweisungen einen Rekonfigurationscode, der den Befehlssatz identifiziert, der dem Untersatz von Befehlsanweisungen entspricht;
b) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um mindestens eine zweite Objektdatei

zu erzeugen; und

c) Geräte für computerlesbaren Programmcode, die konfiguriert sind, um die erzeugten Objektdateien zu binden, die erzeugt wurden, um eine ausführbare Datei zu erzeugen.

33. Computerprogrammprodukt nach Anspruch 32, das außerdem Geräte für computerlesbaren Programmcode umfaßt, die konfiguriert sind, um auf der erzeugten ausführbaren Datei entsprechend den Ausrichtungserfordernissen eine Speicherausrichtung vorzunehmen.

Hierzu 37 Seite(n) Zeichnungen

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

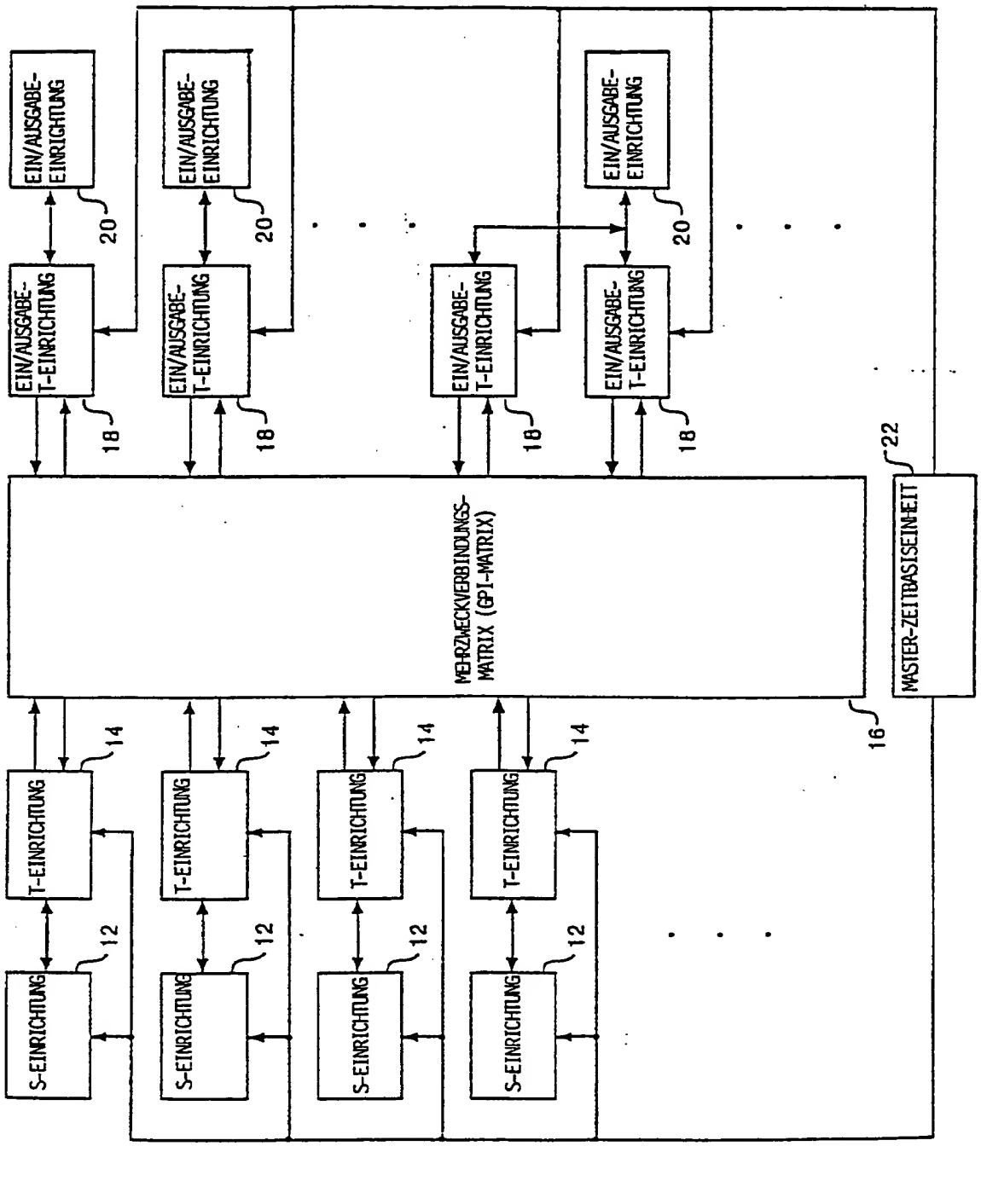
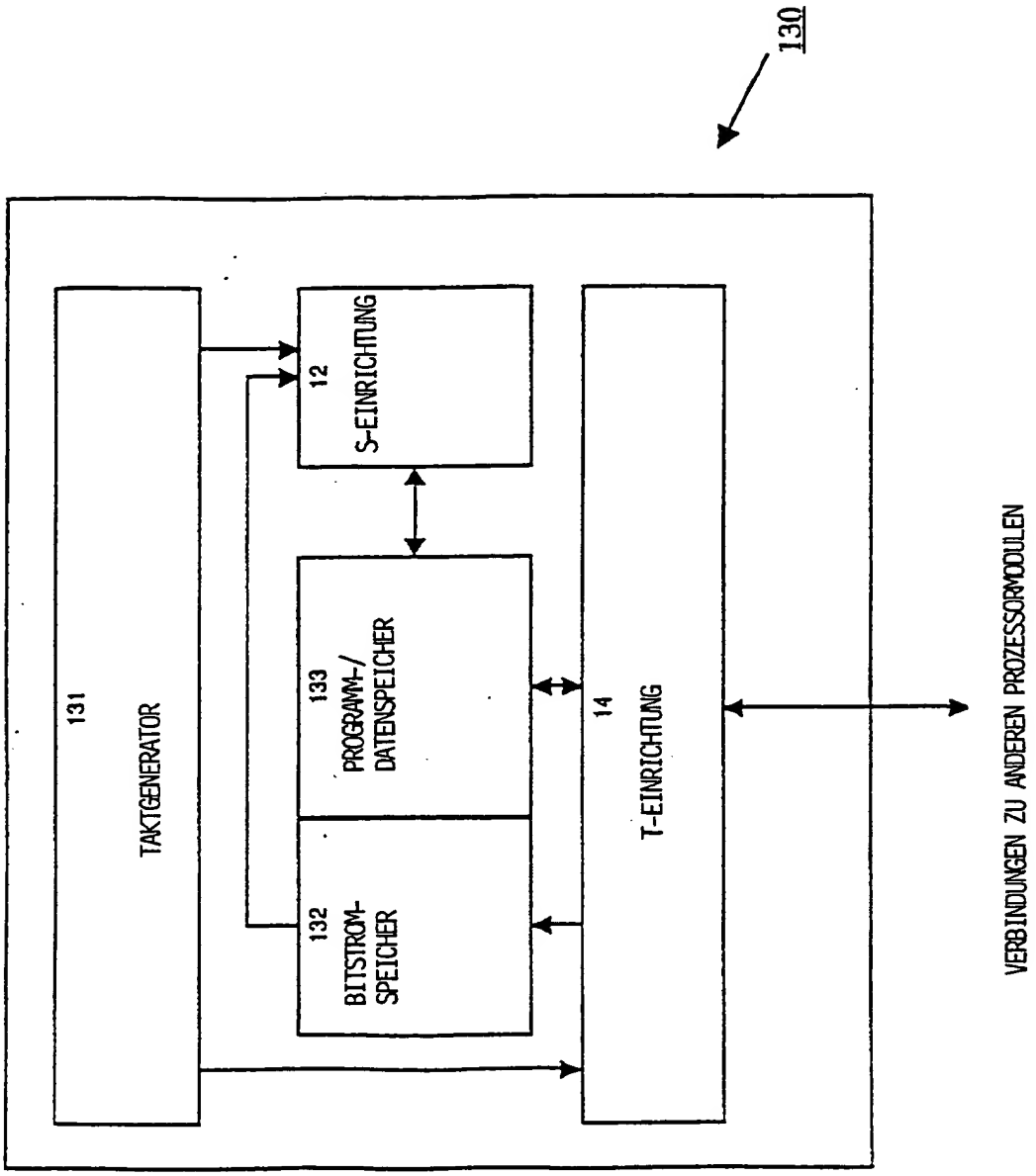


FIG. 1

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



FIGUR 1A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

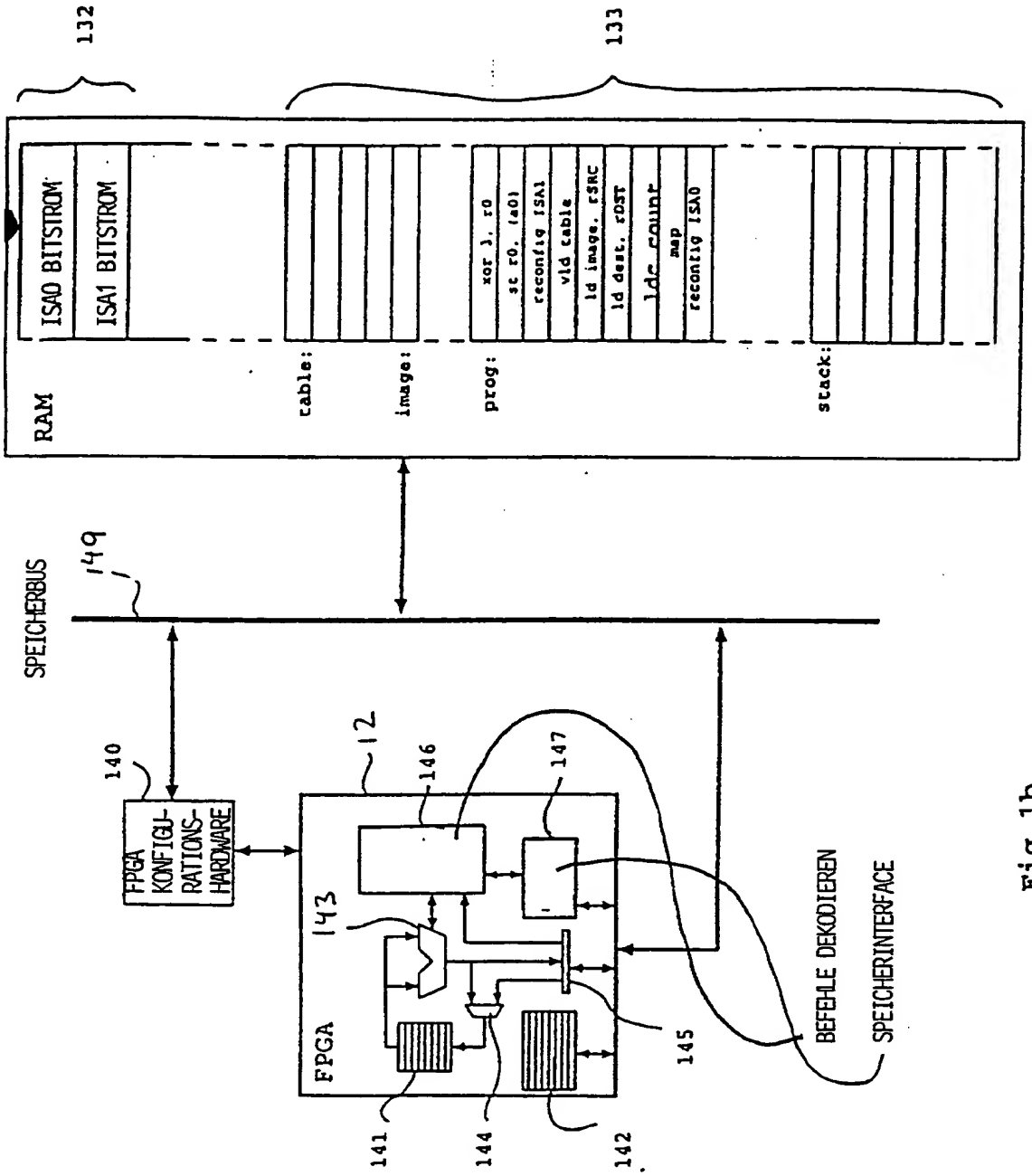


Fig 1b

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

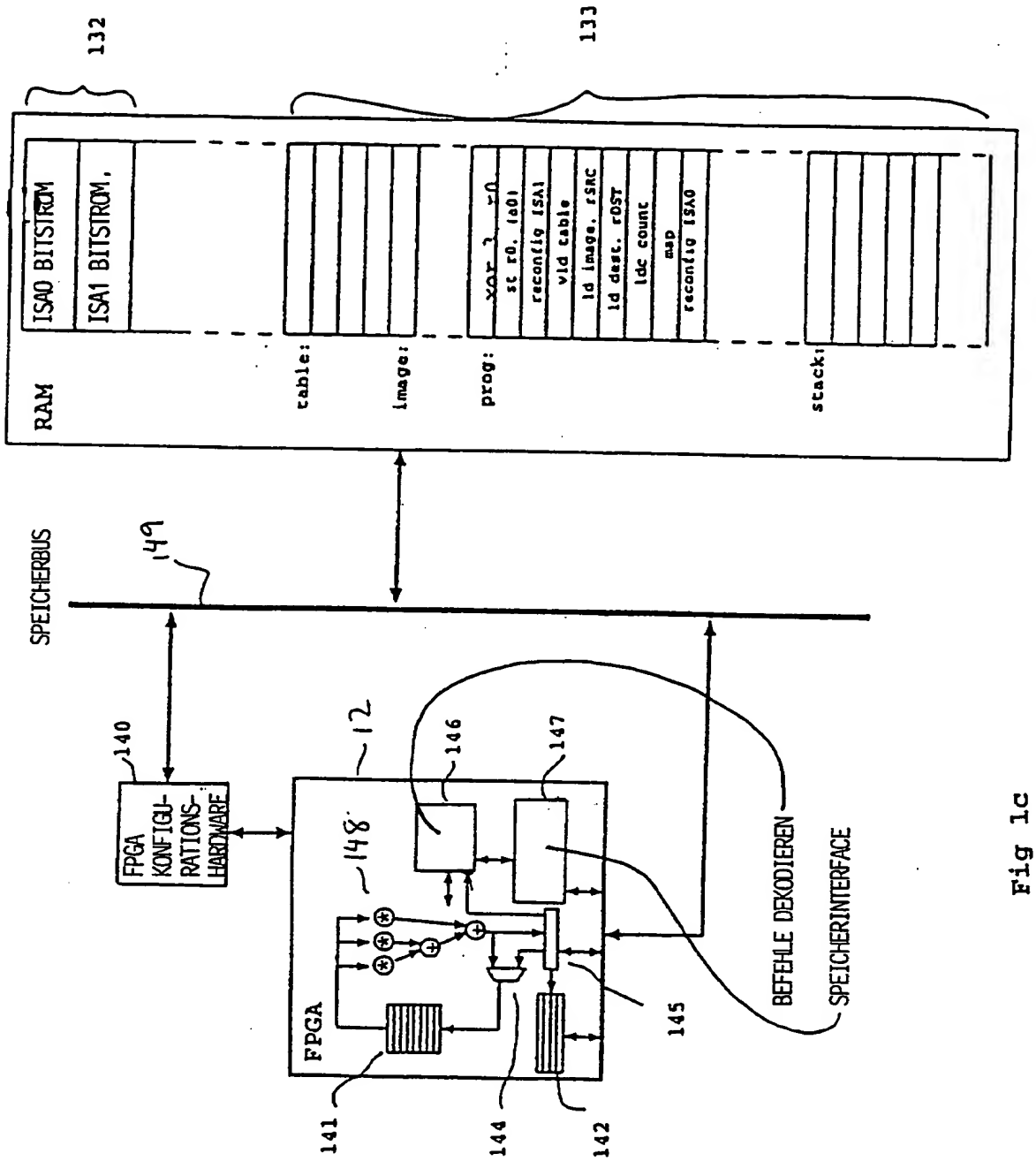
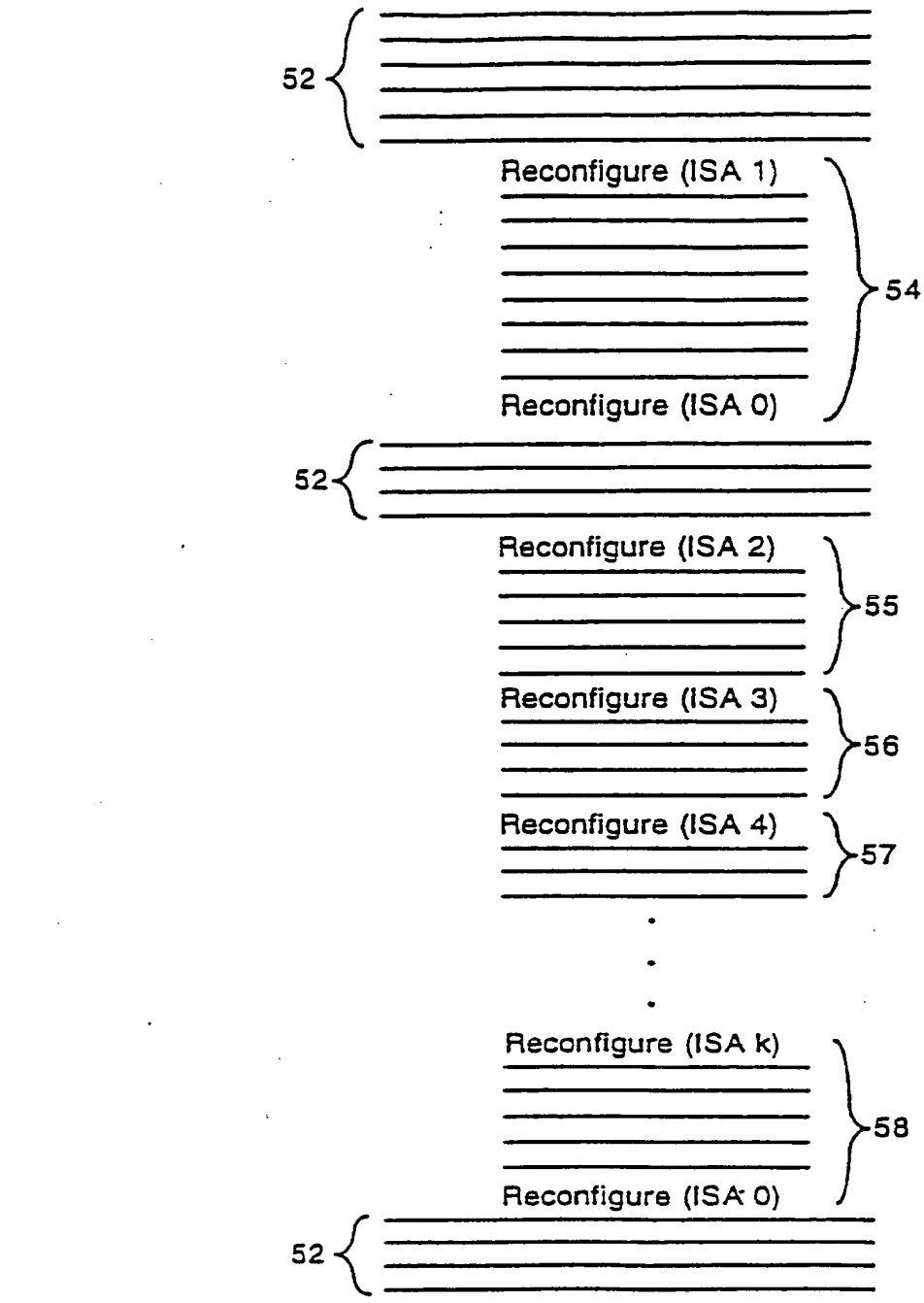


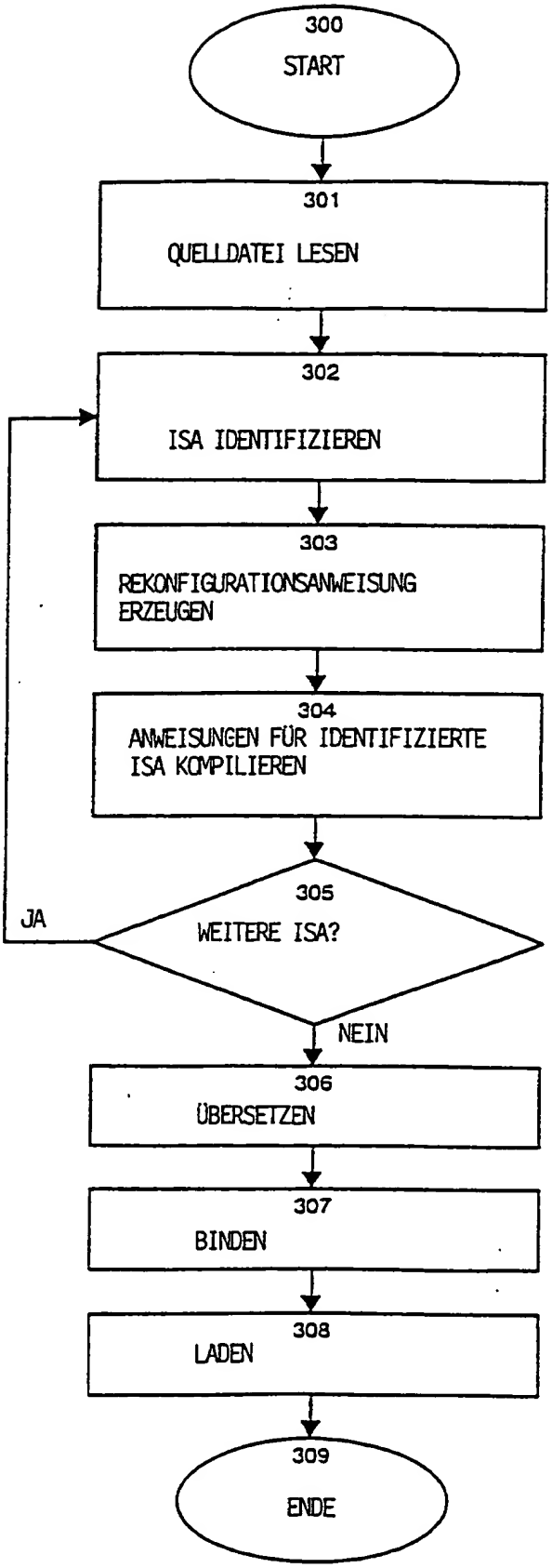
Fig 1c

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



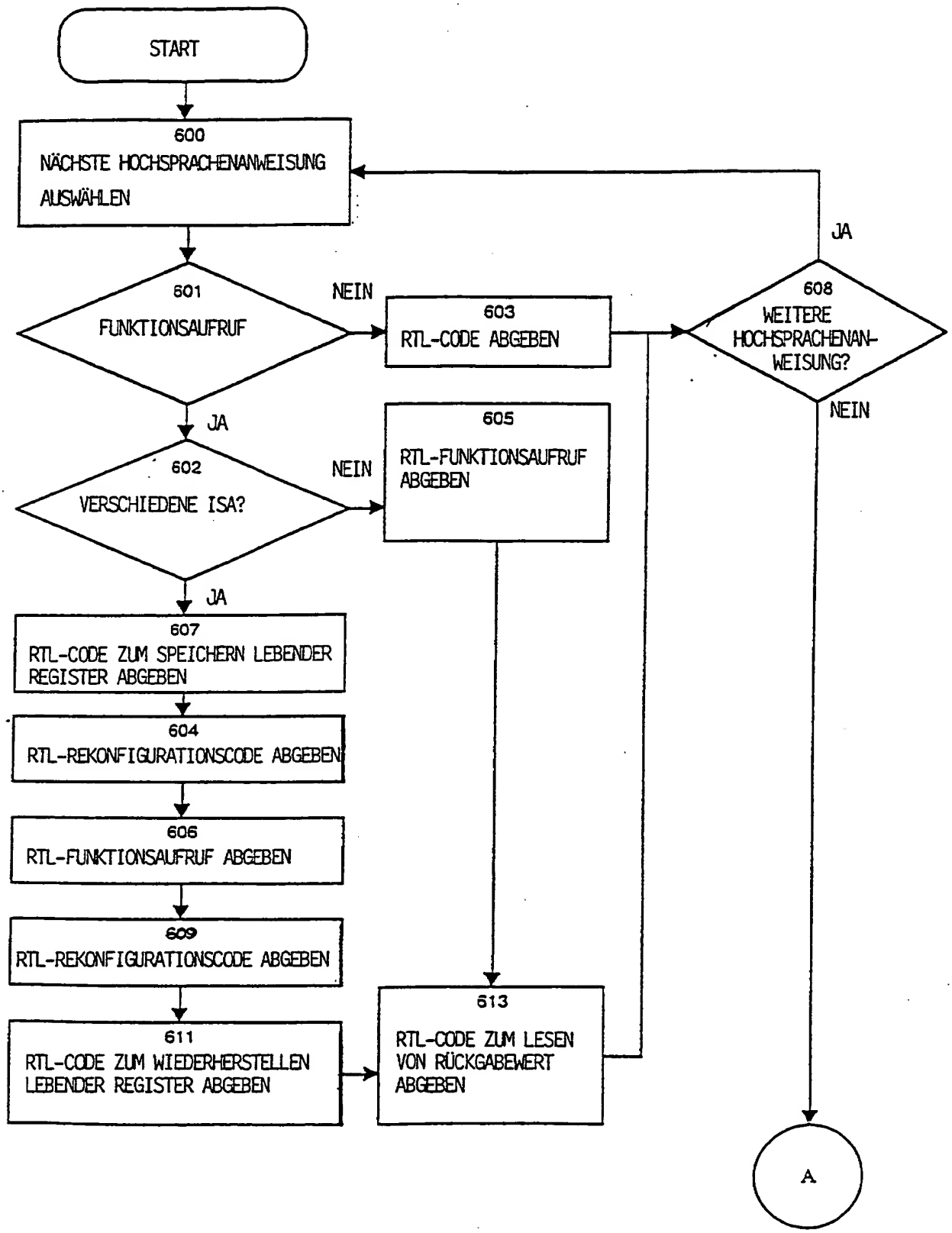
FIGUR 2

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



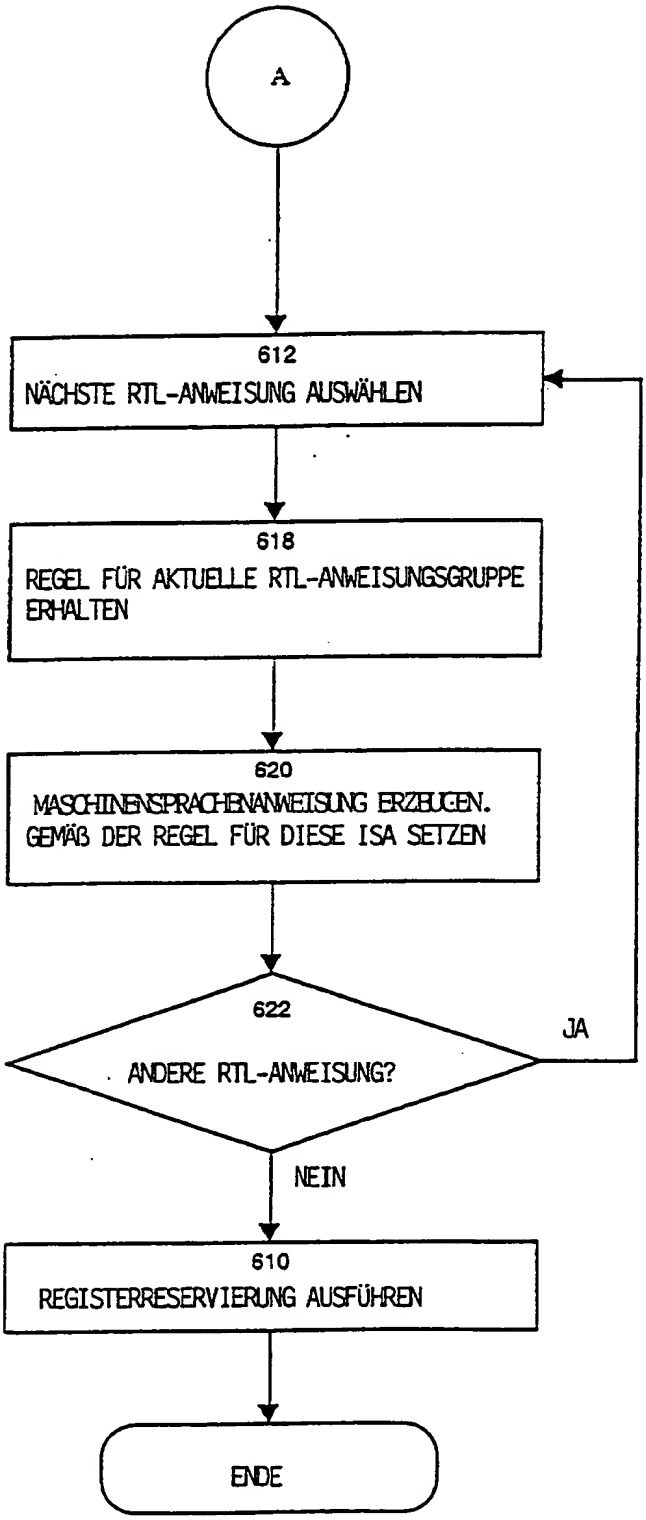
FIGUR 3

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



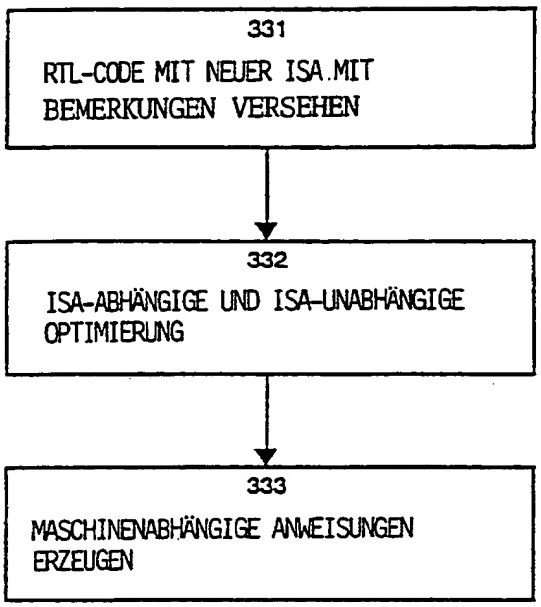
FIGUR 3A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



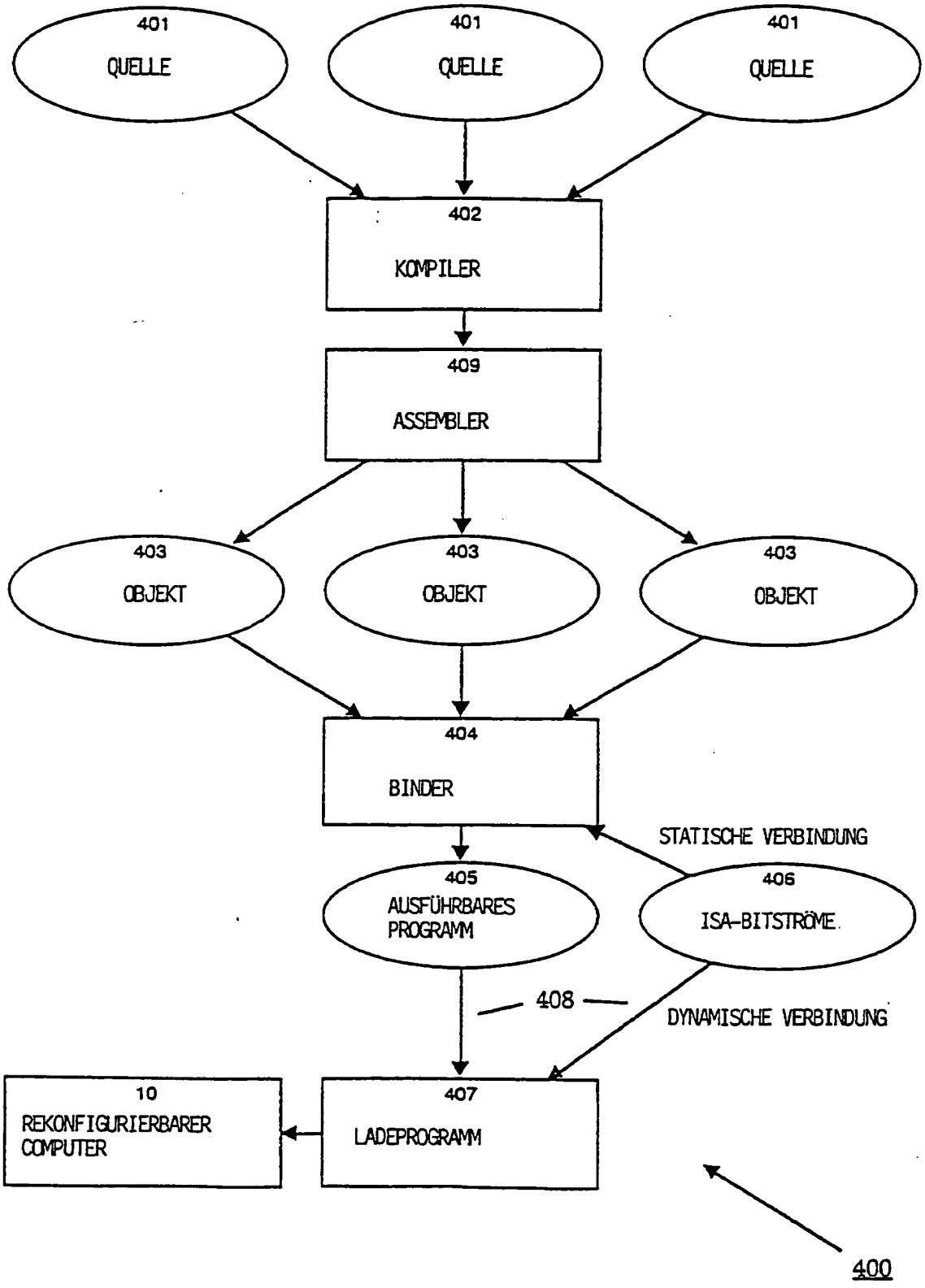
FIGUR 3B

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



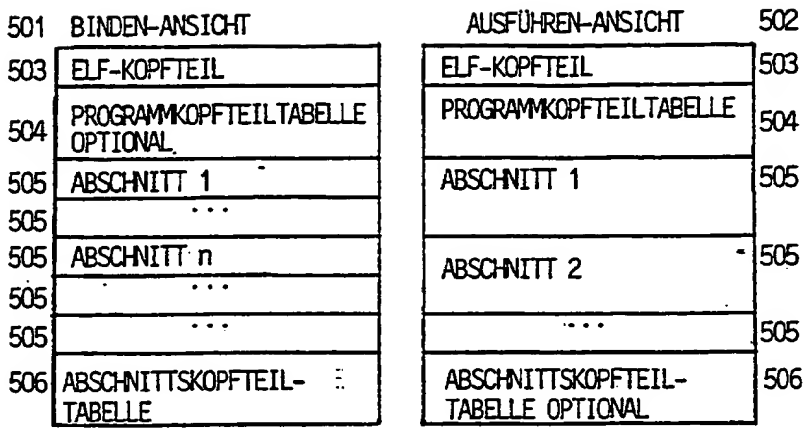
FIGUR 3C

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



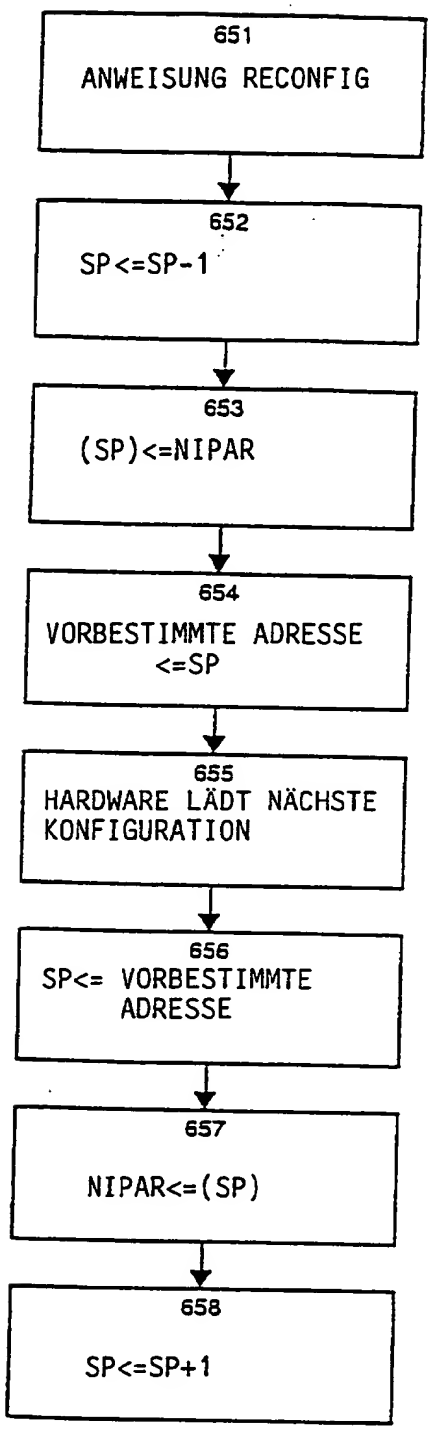
FIGUR 4

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



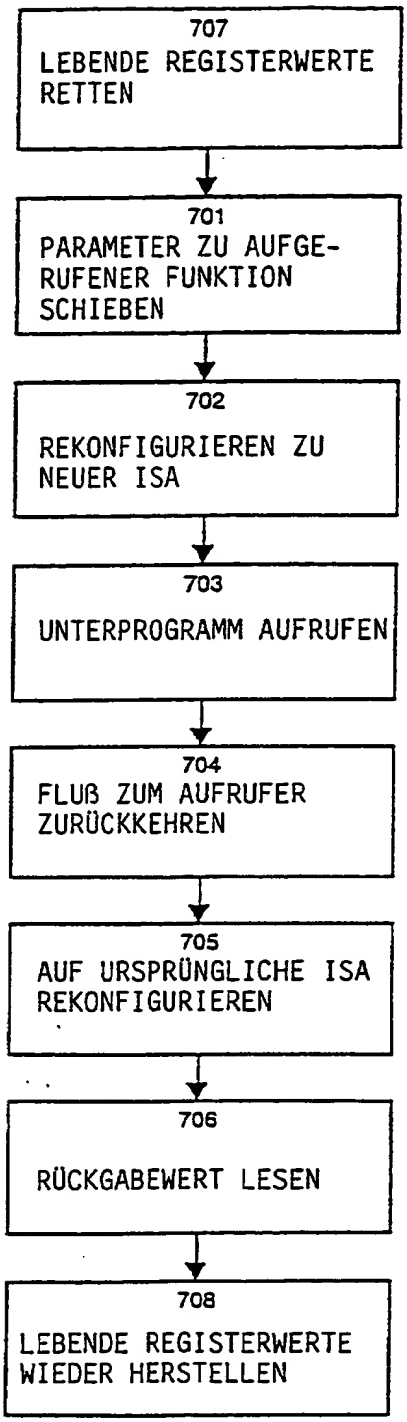
FIGUR 5 (STAND DER TECHNIK)

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



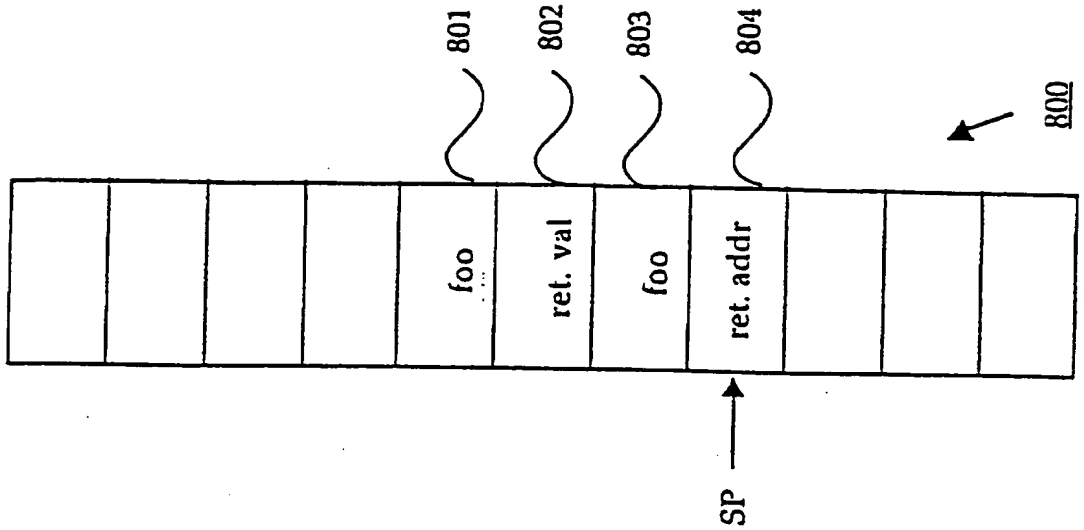
FIGUR 6

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

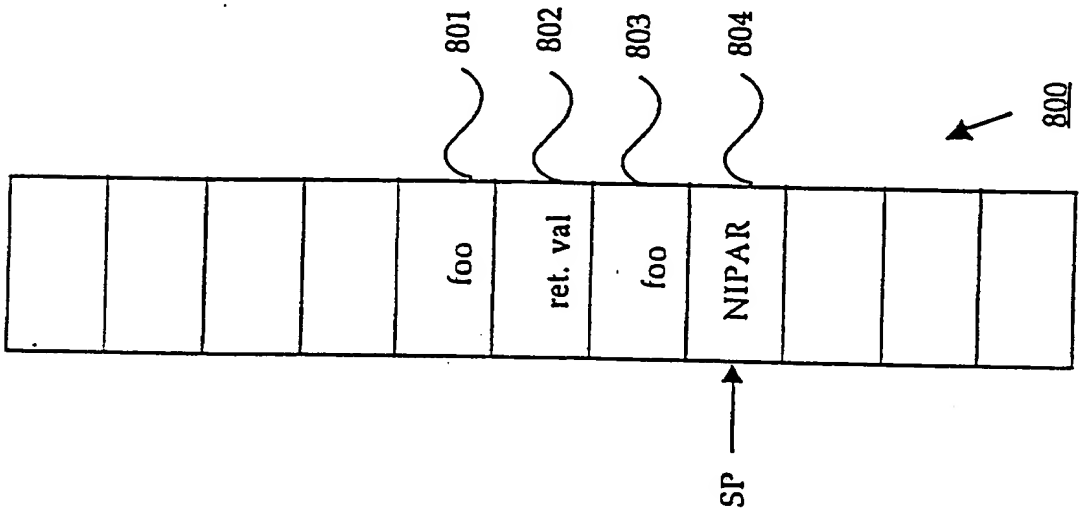


FIGUR 7

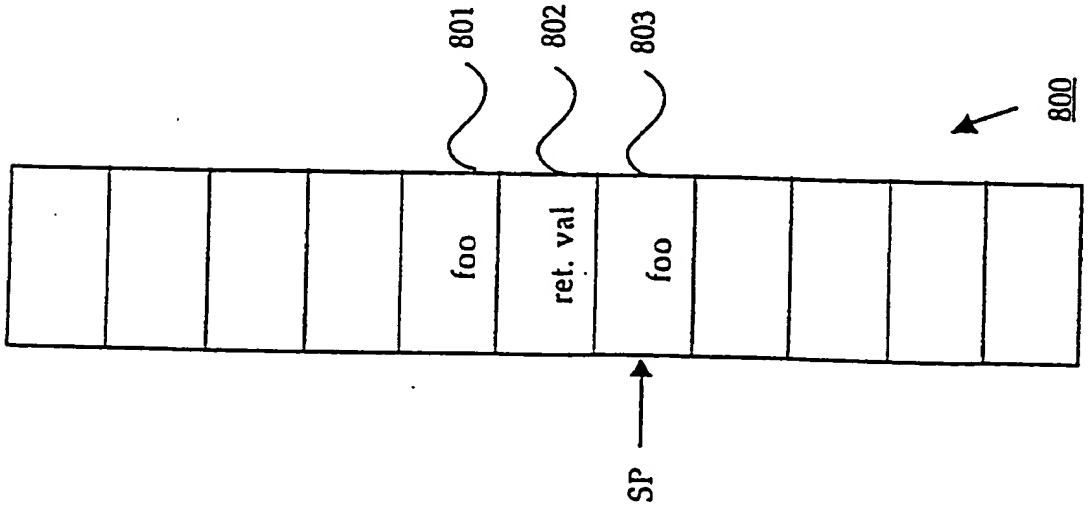
Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998



FIGUR 8C



FIGUR 8B



FIGUR 8A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

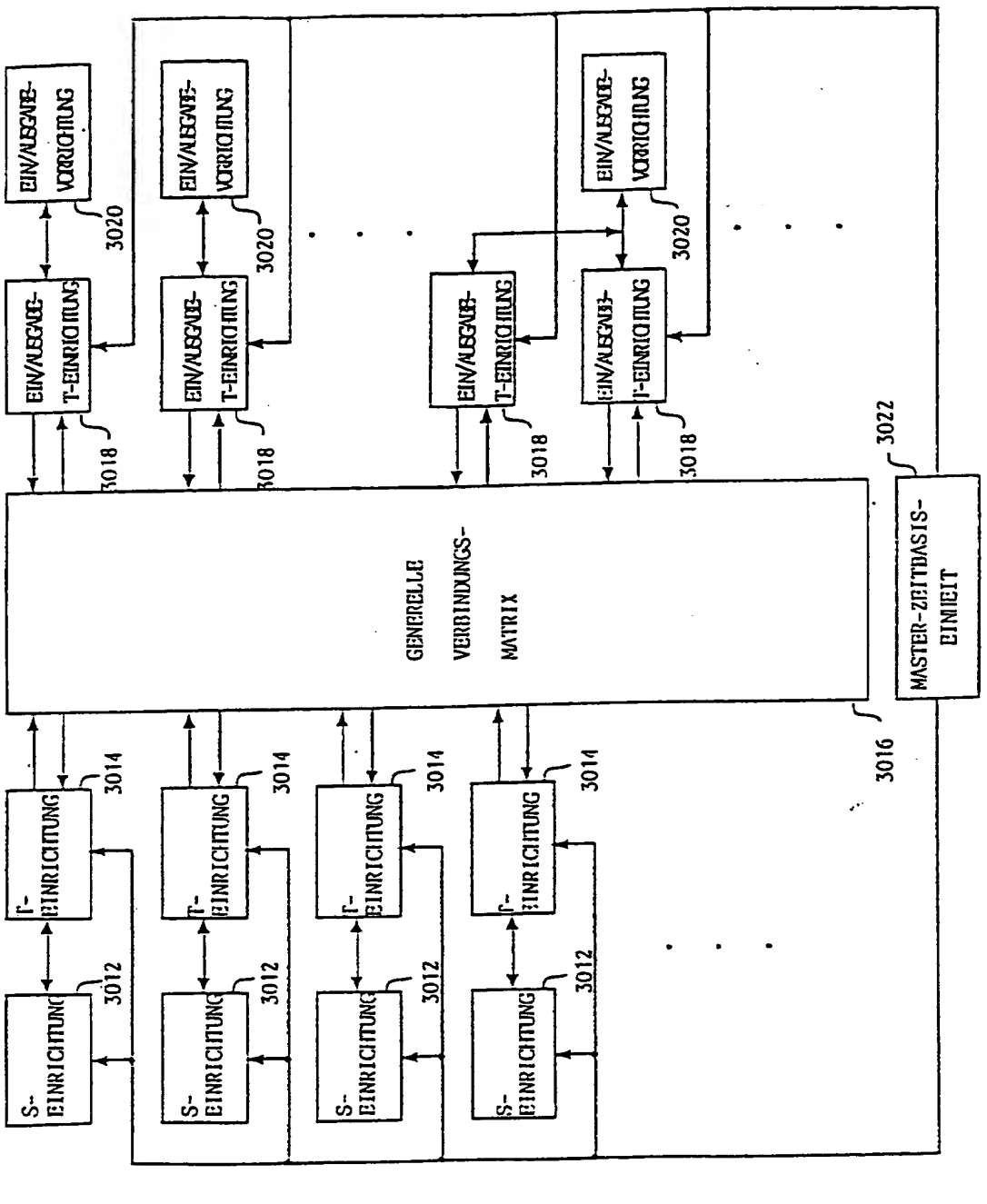


FIG. 9

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

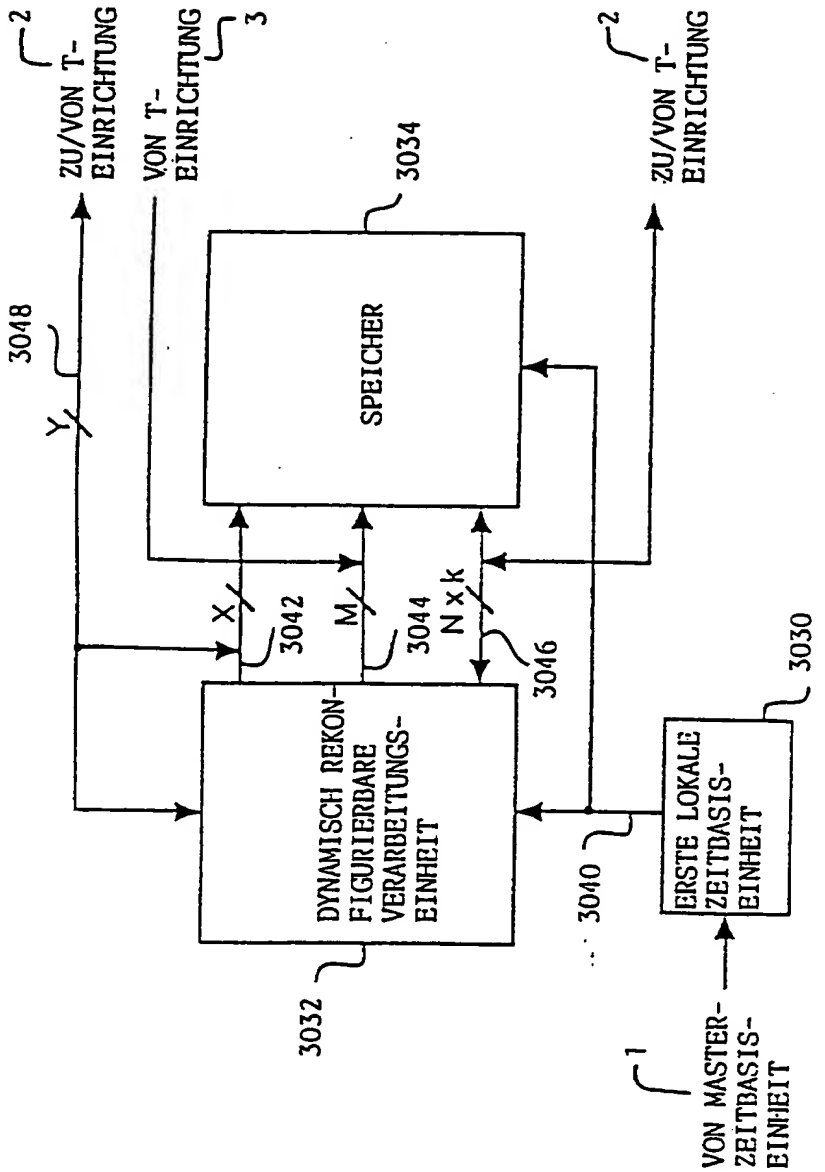


FIG. 10

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

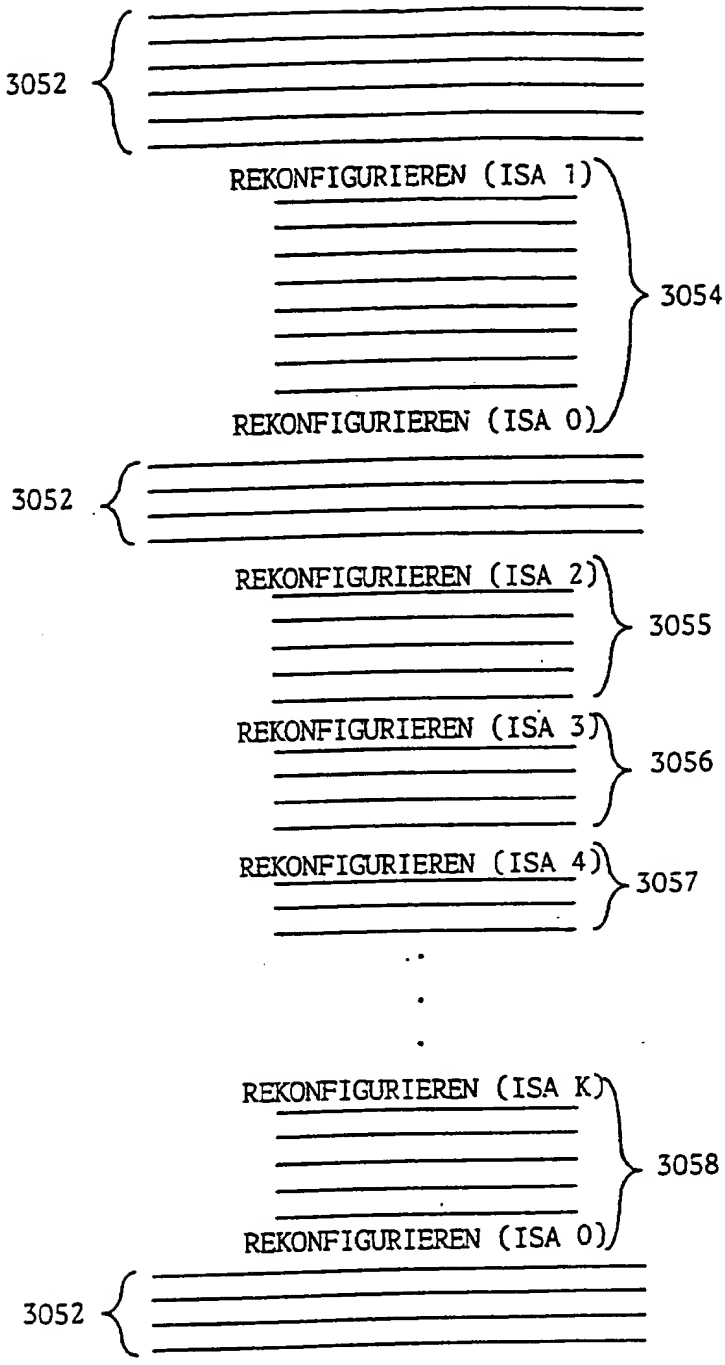


FIG. 11 A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

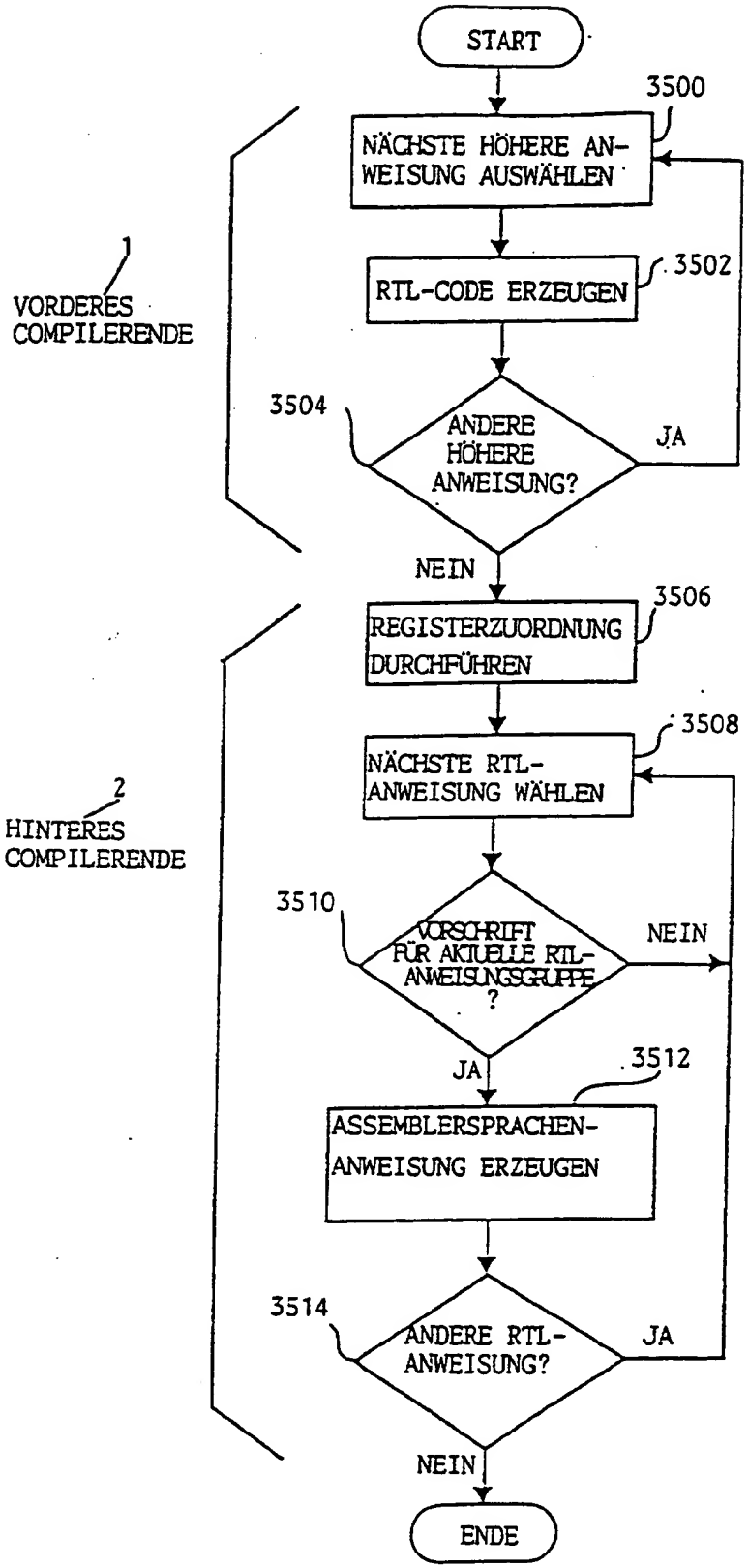


FIG. 11 B

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

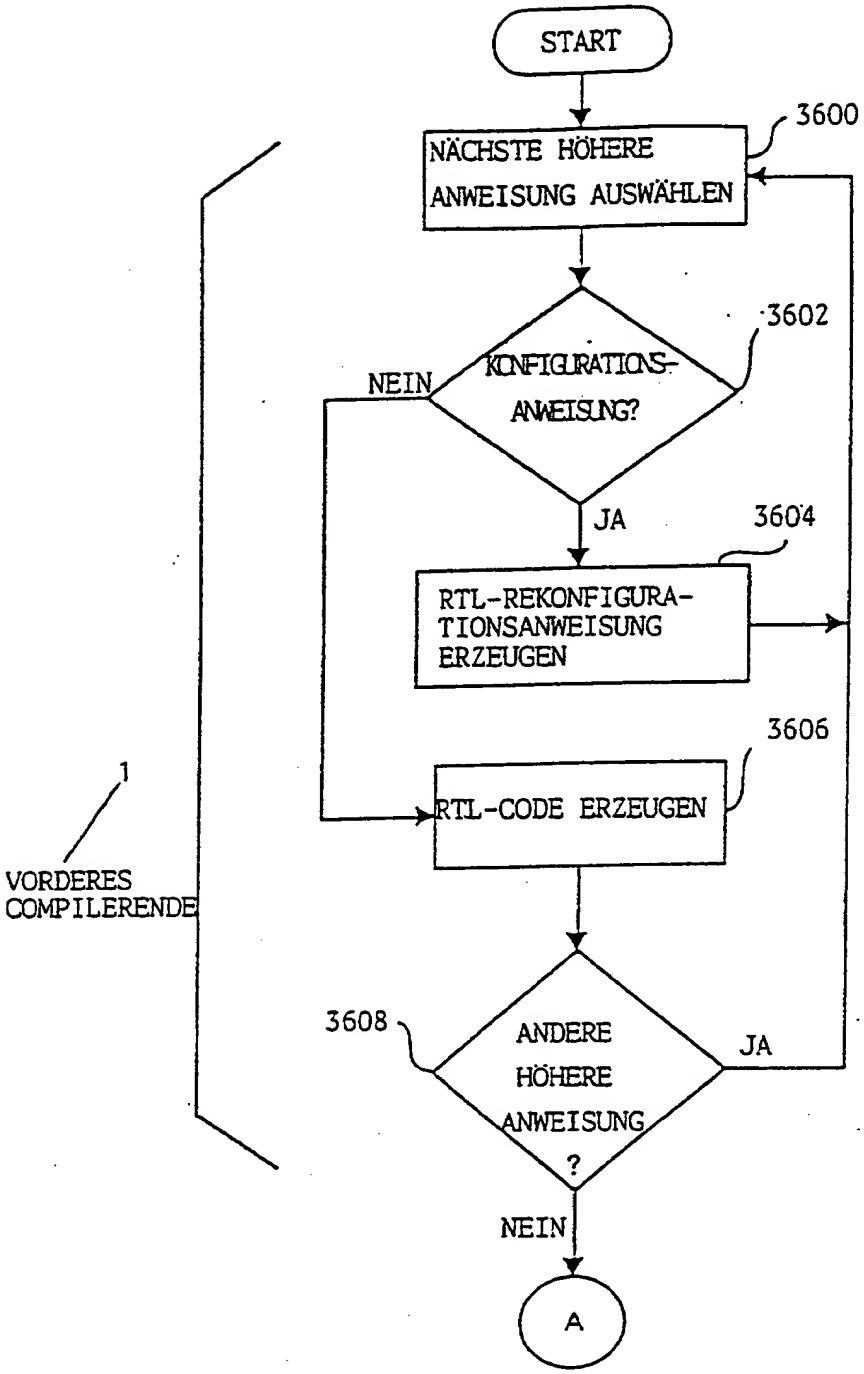


FIG. 11C

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

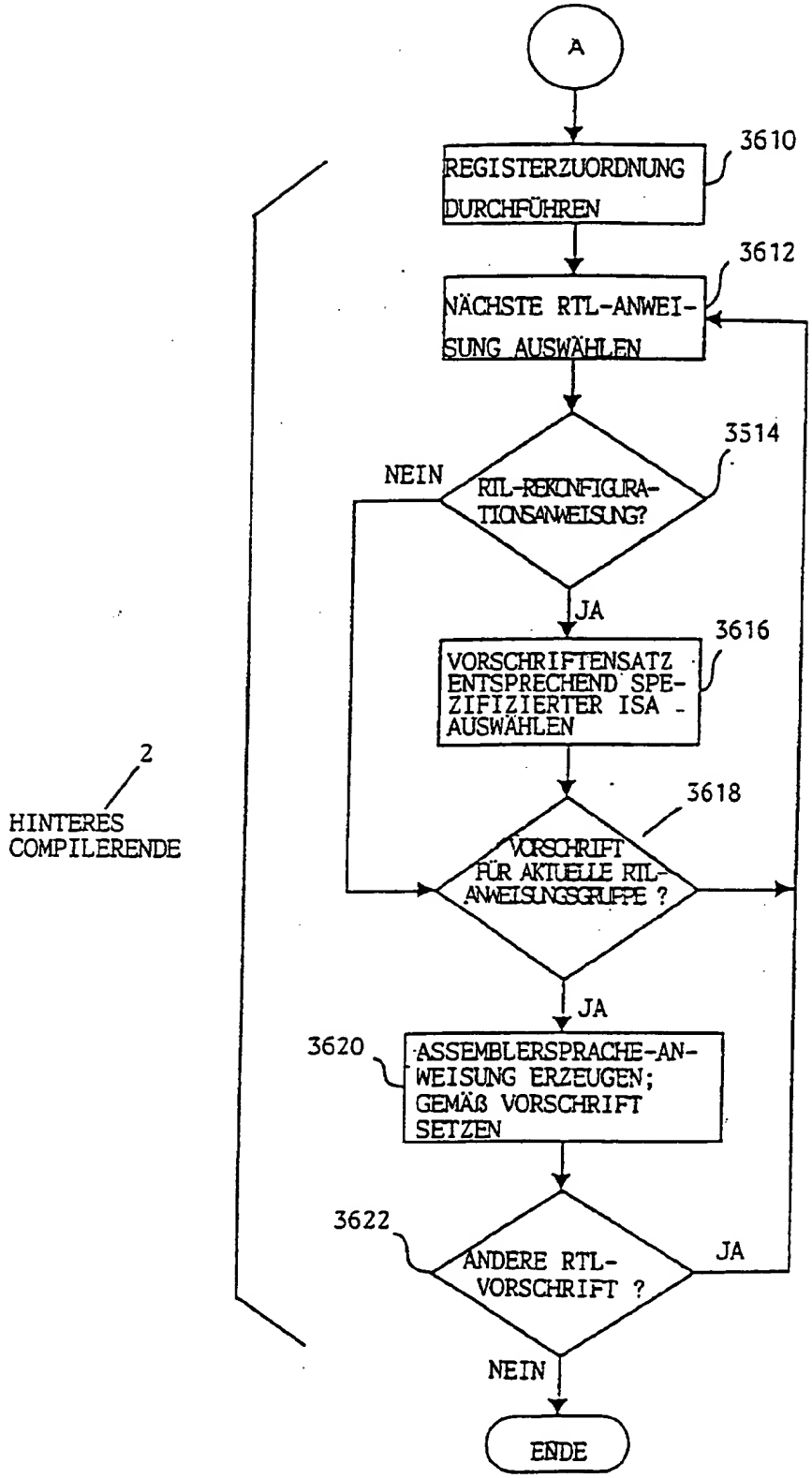


FIG. 11 D

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

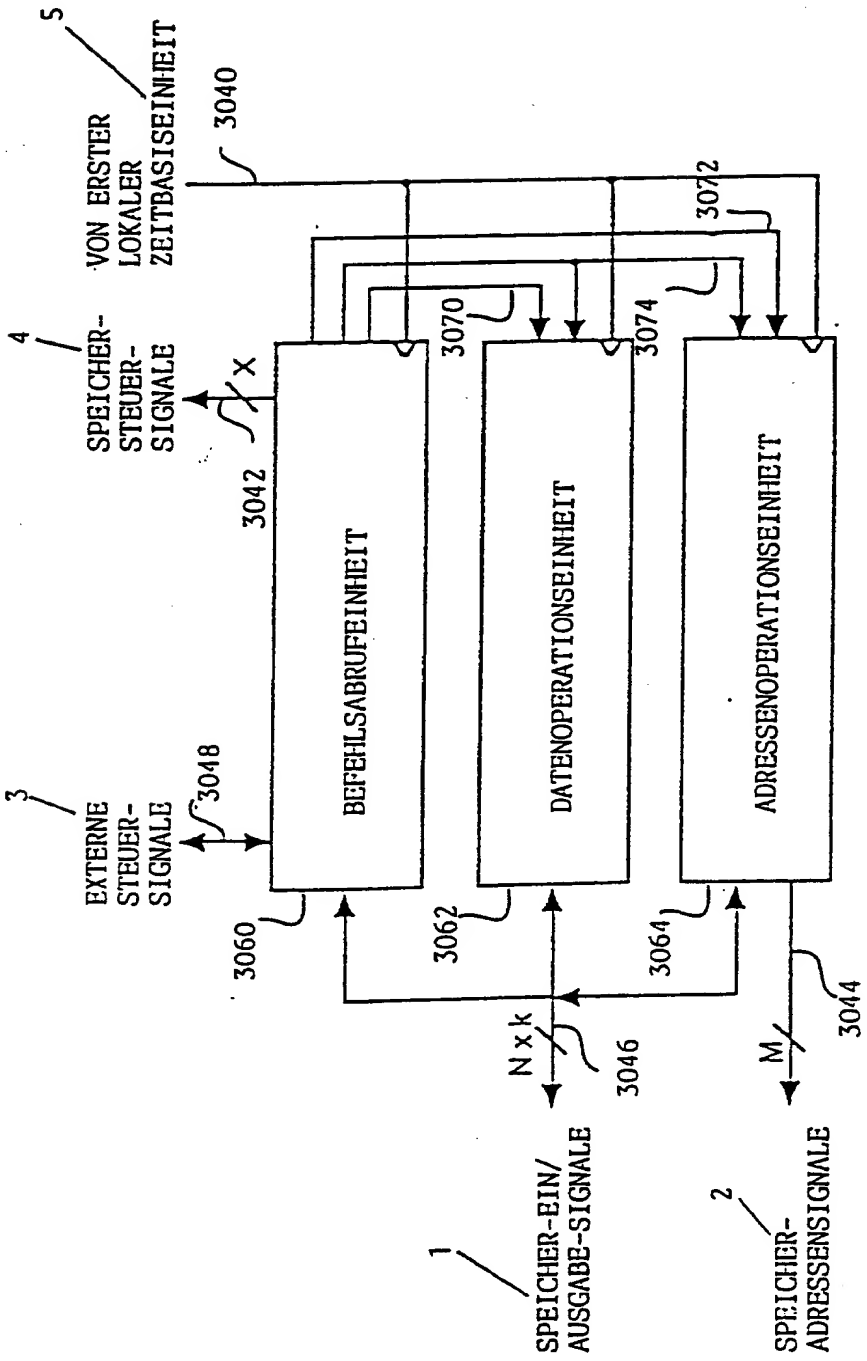


FIG. 12

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

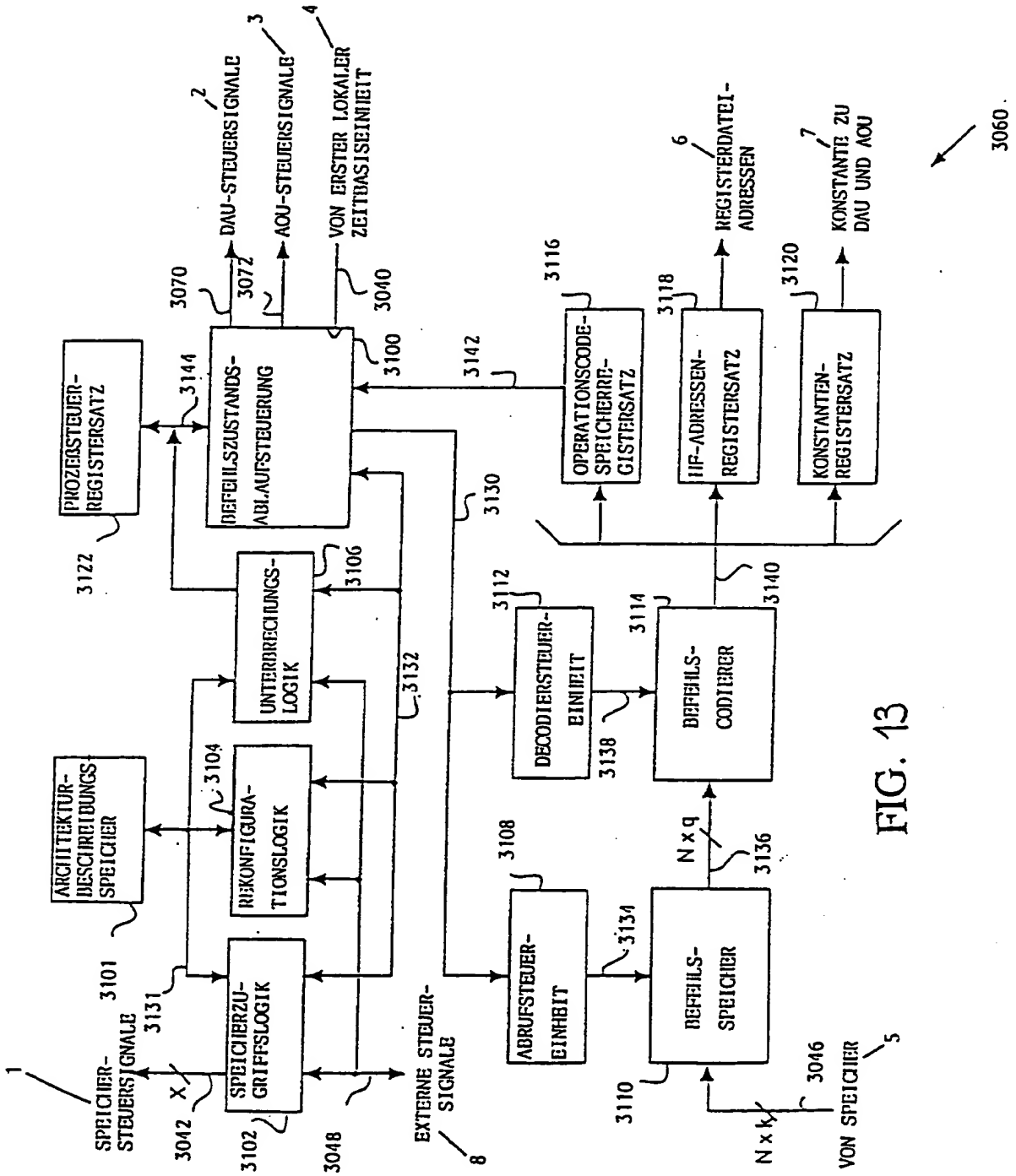


FIG. 13

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

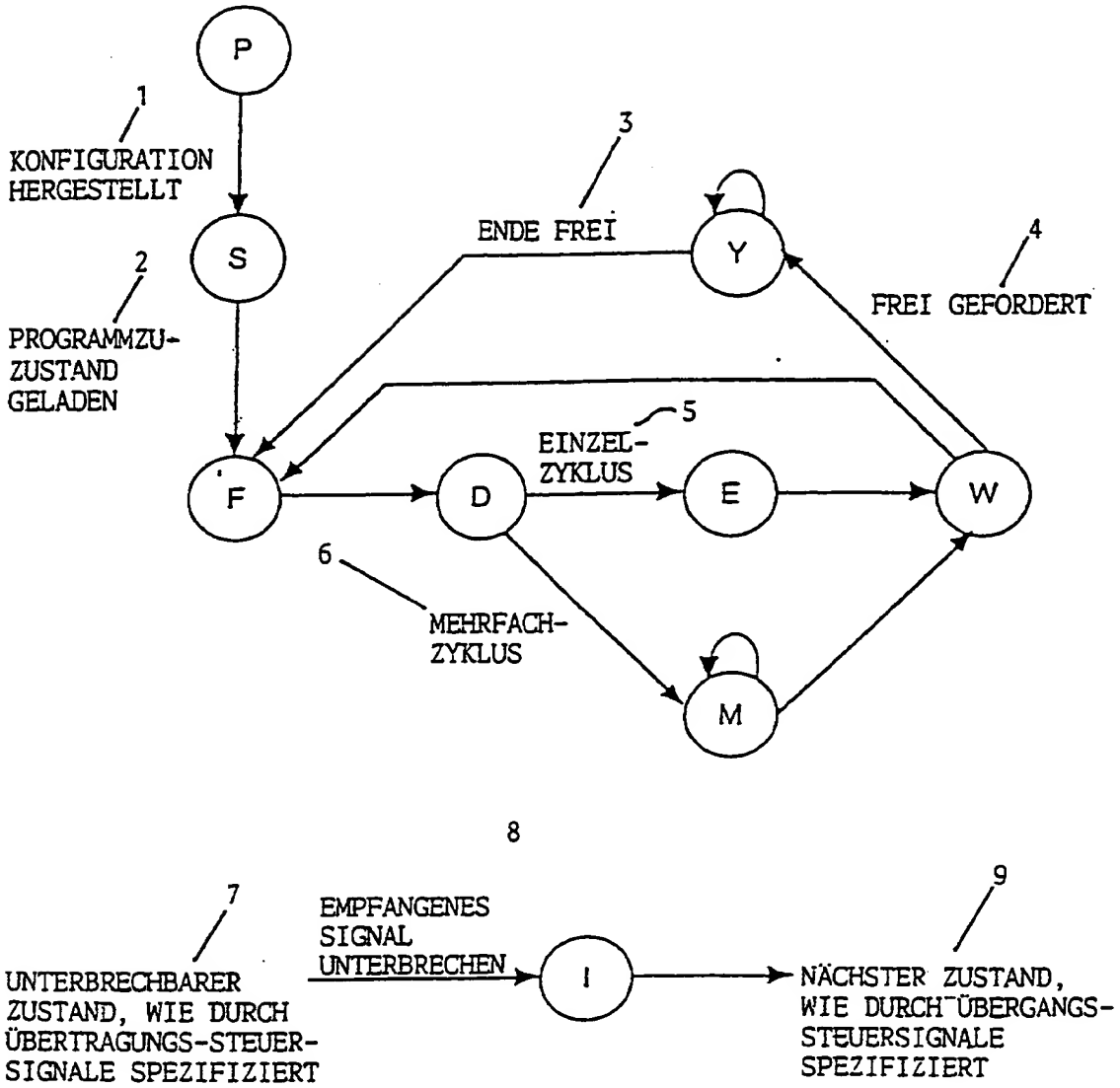


FIG. 14

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

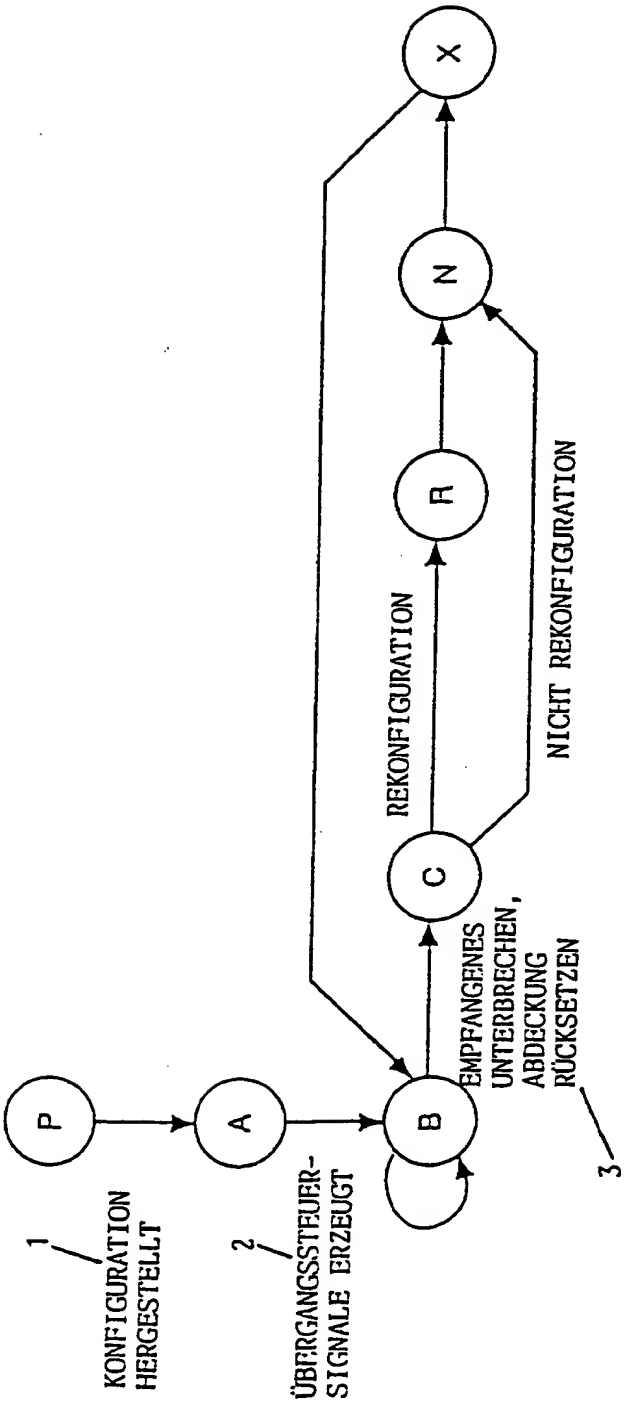


FIG. 15

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

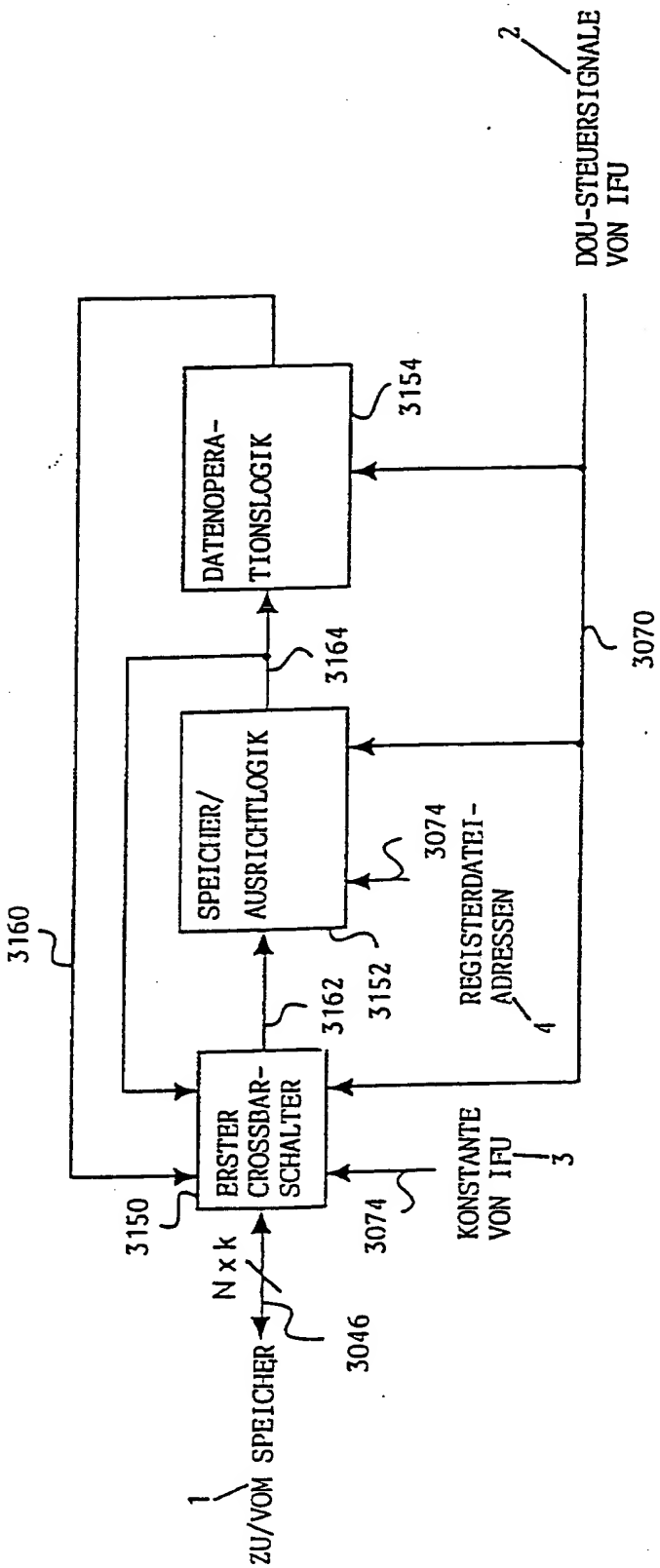


FIG. 16

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

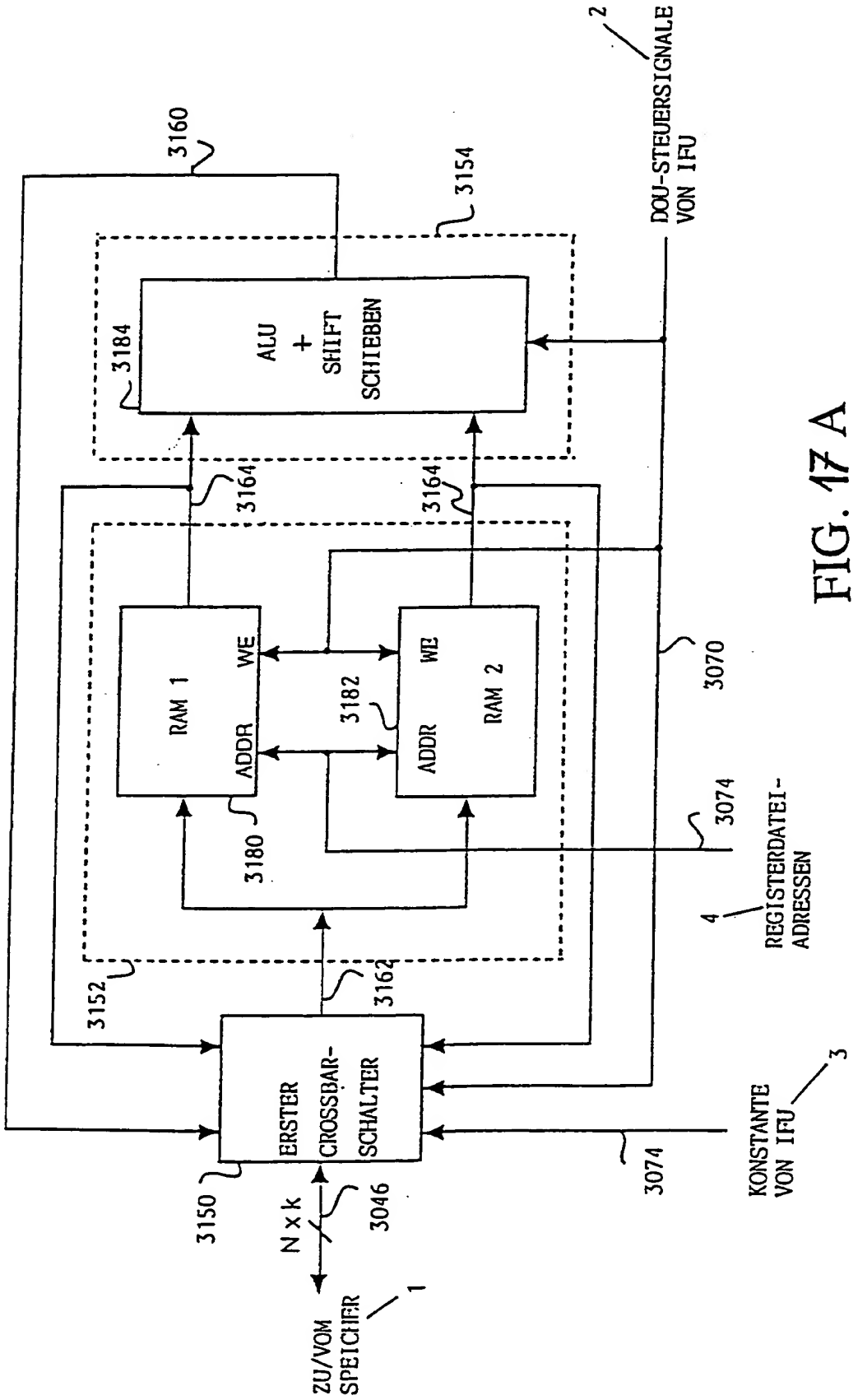


FIG. 17 A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

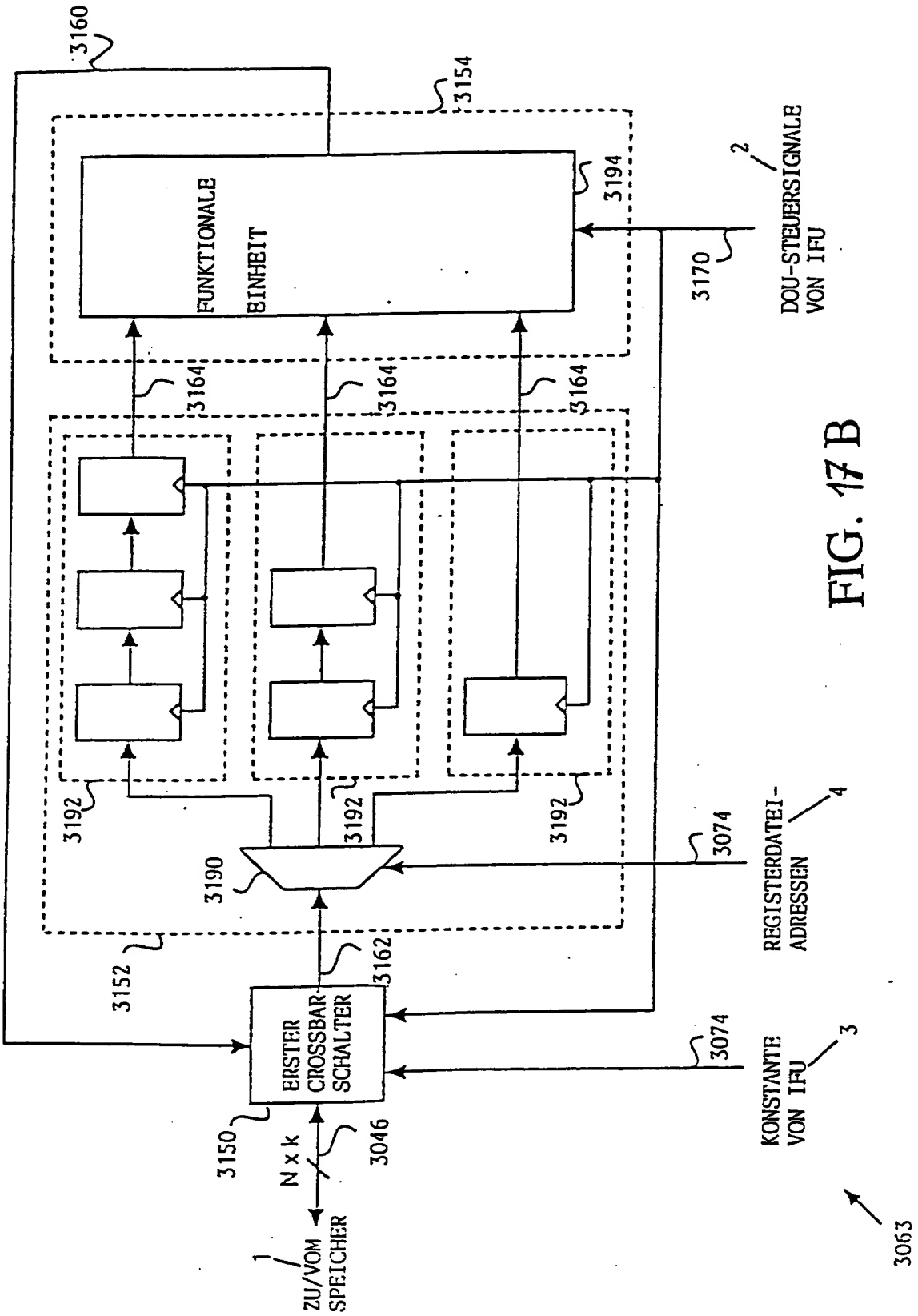


FIG. 17 B

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

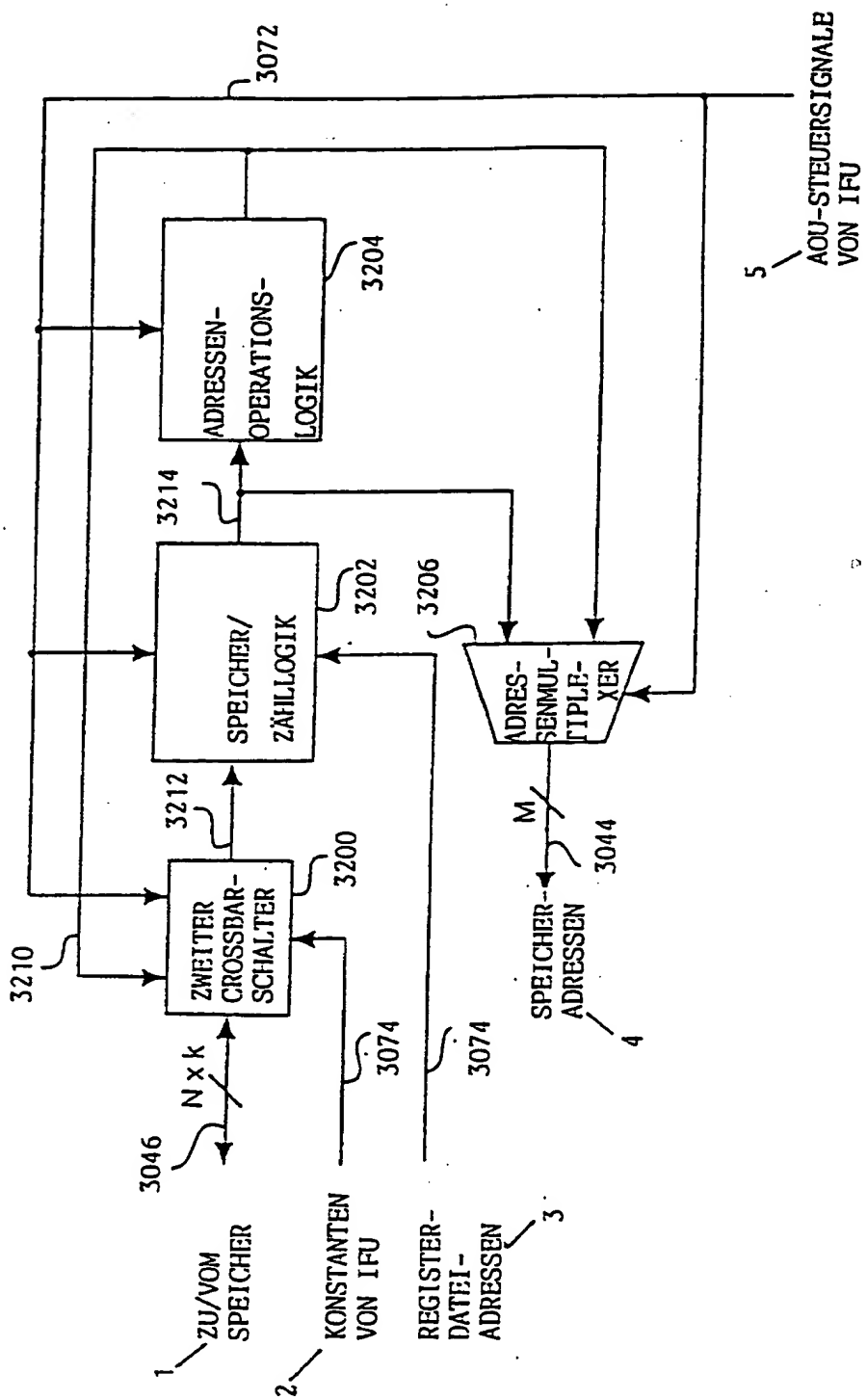


FIG. 18

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

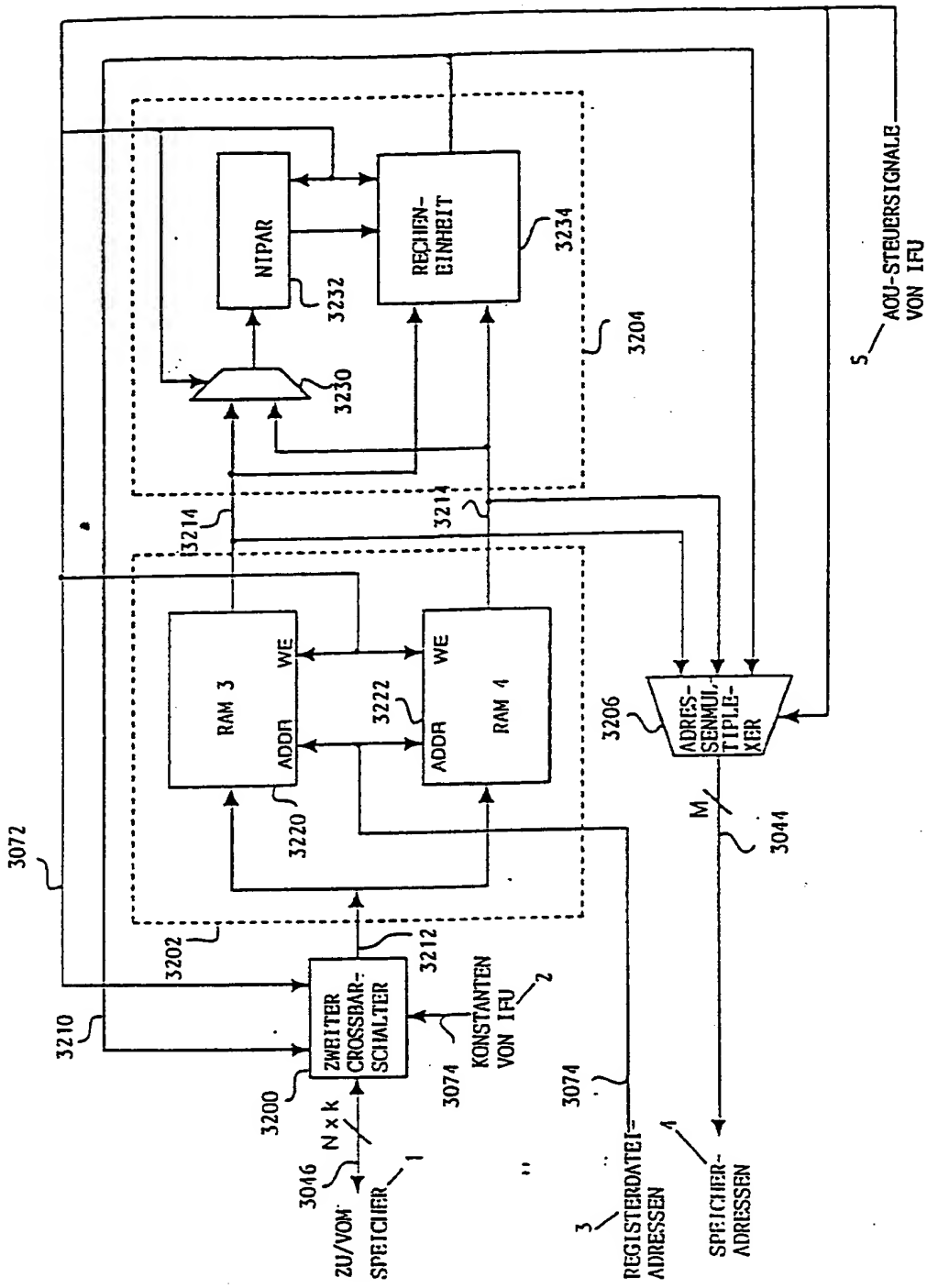


FIG. 19A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

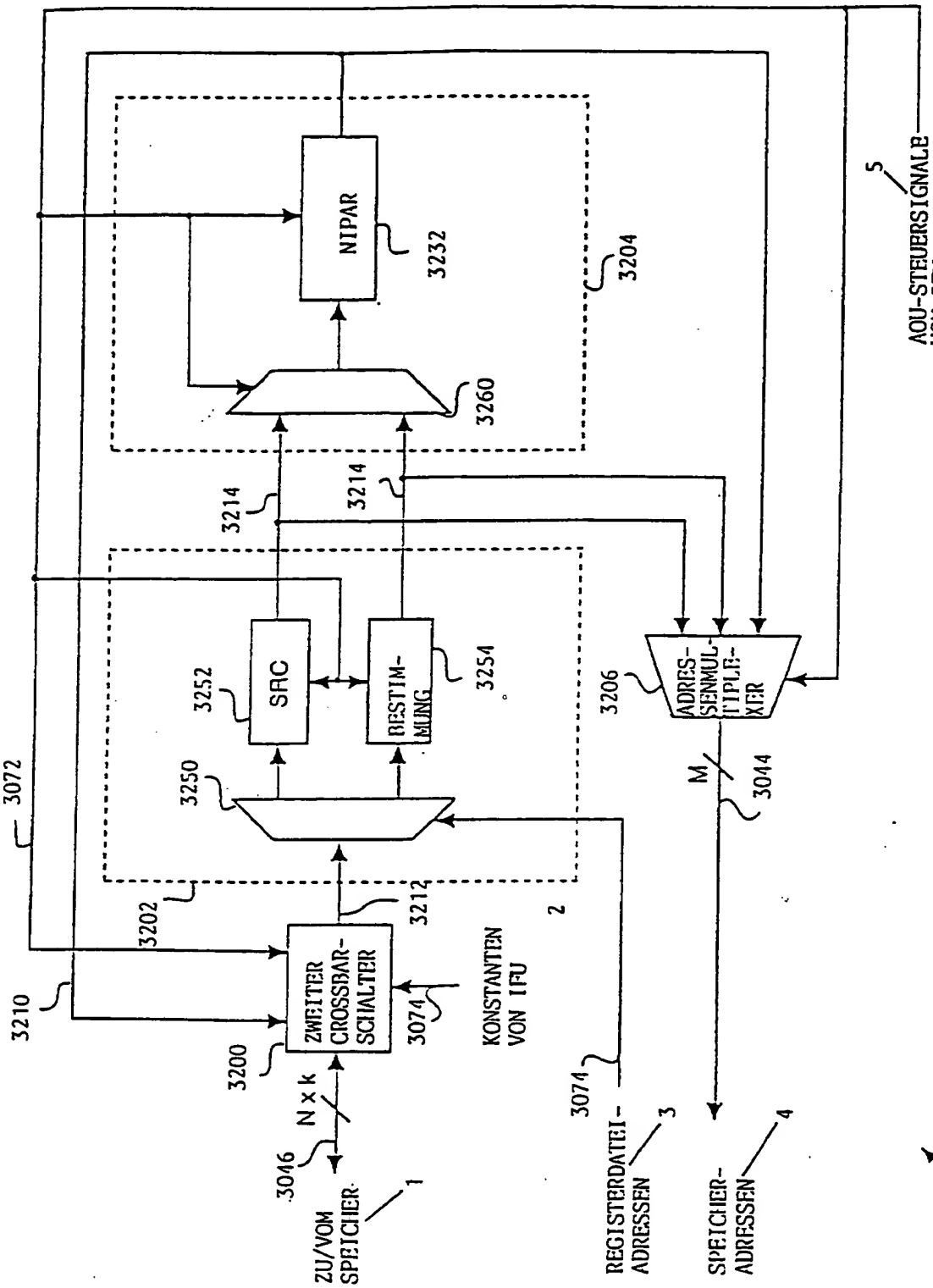


FIG. 19B

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

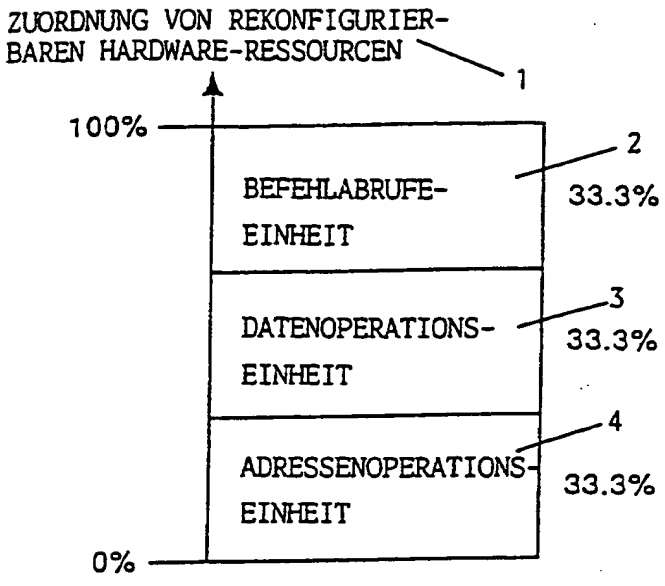


FIG. 20A

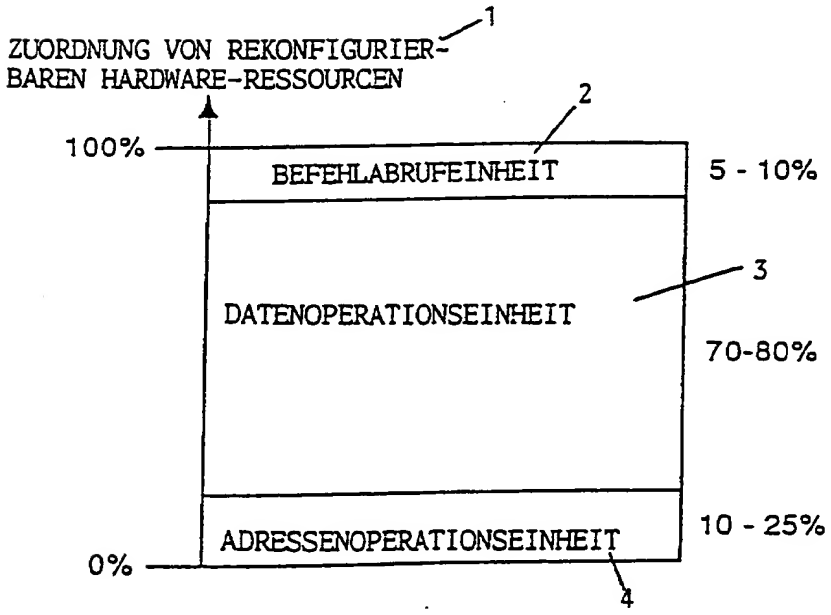


FIG. 20B

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

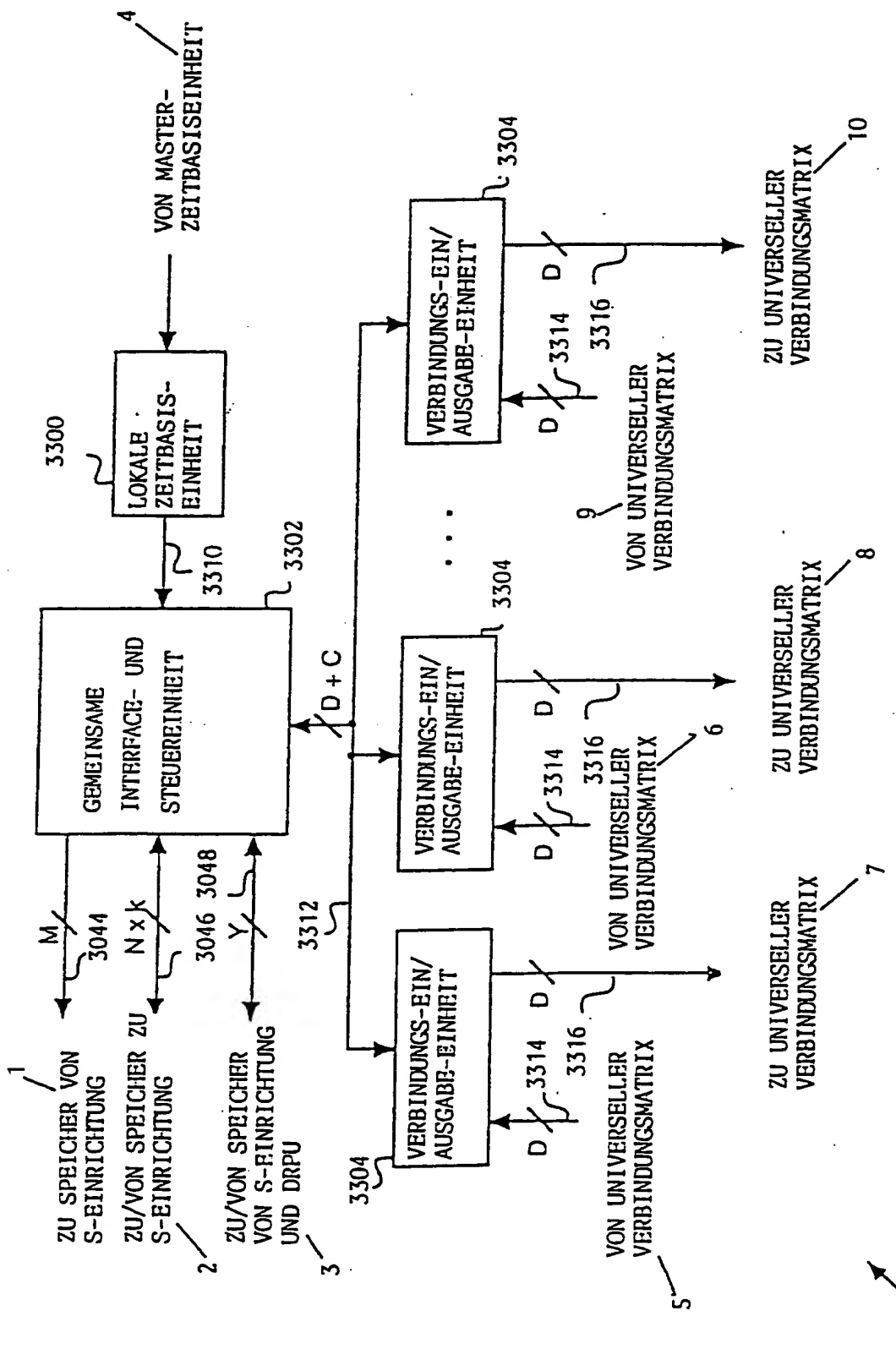


FIG. 21

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

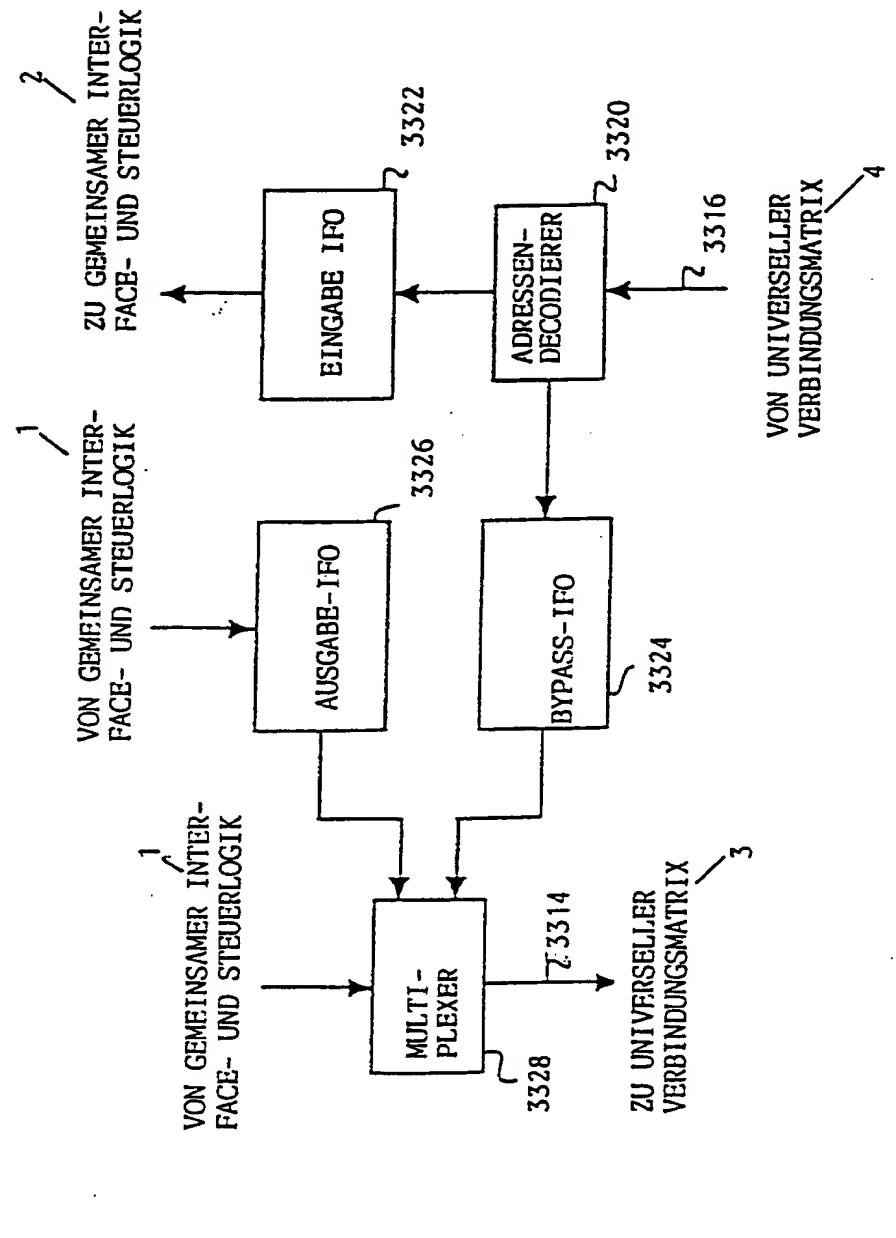


FIG. 22

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

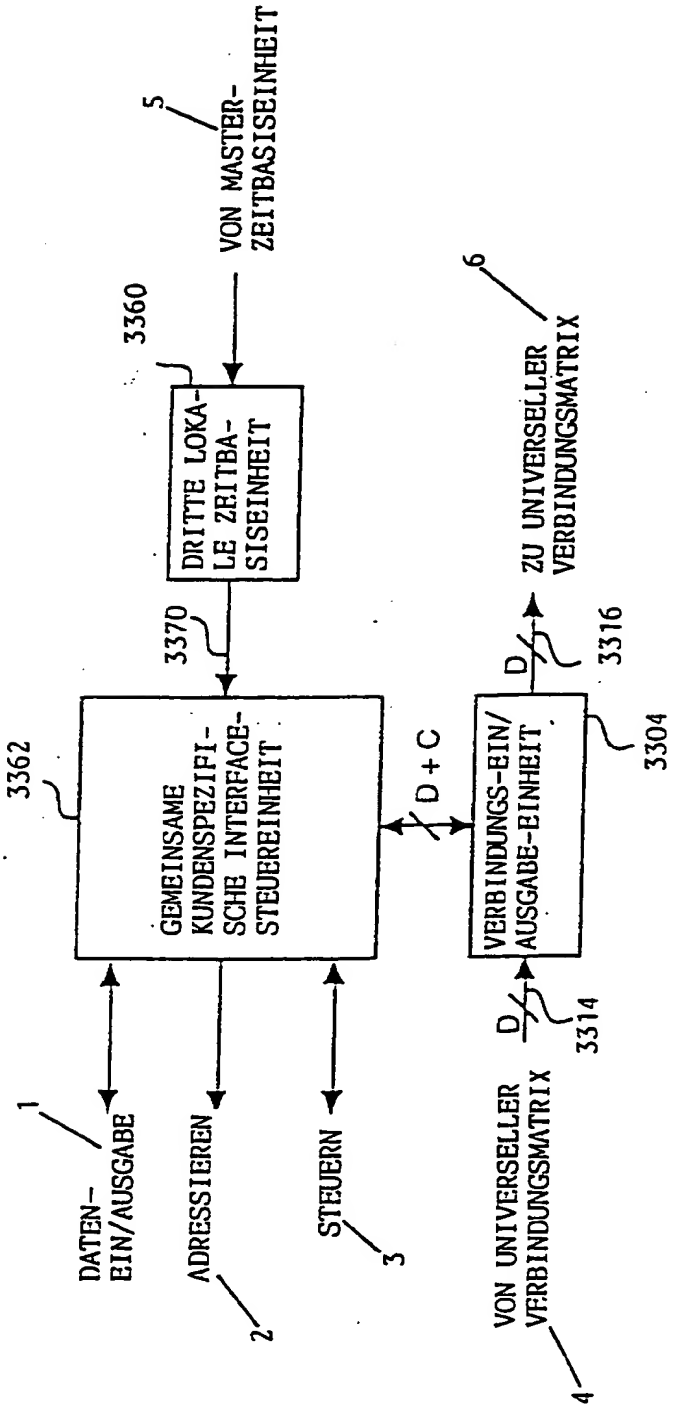


FIG. 23

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

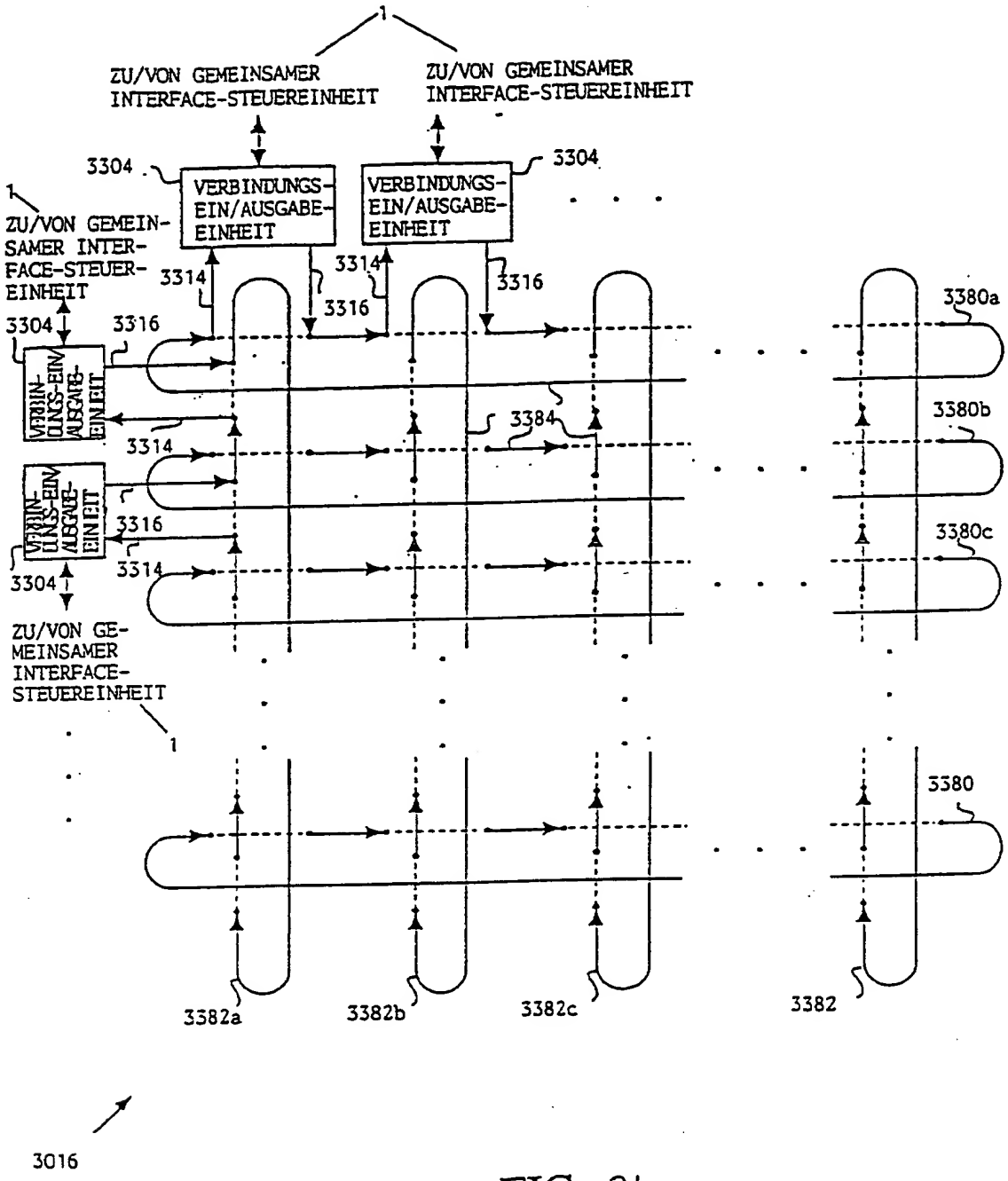


FIG. 24

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

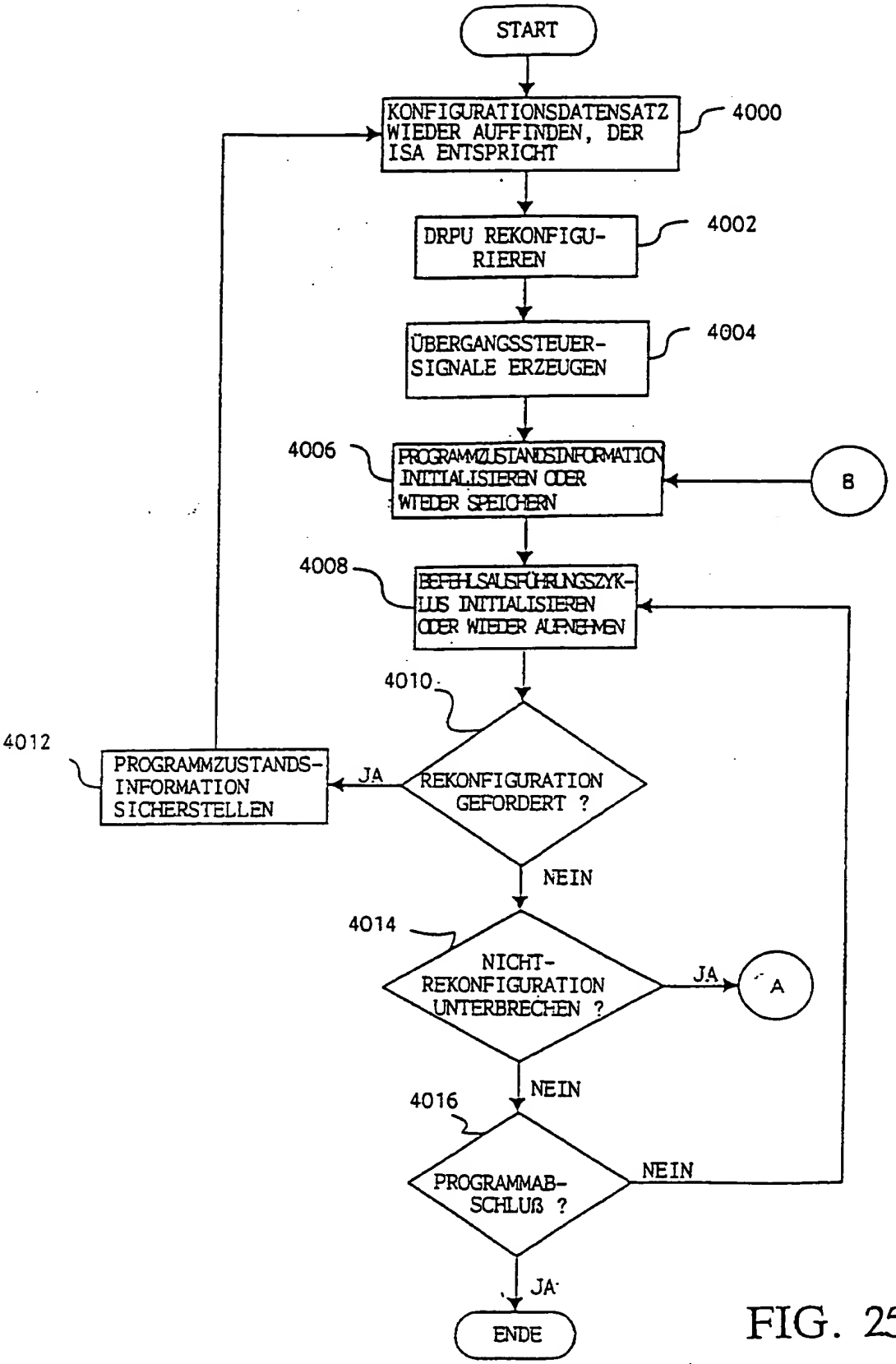


FIG. 25A

Nummer: DE 198 15 865 A1
Int. Cl.[6]: G 06 F 9/45
Offenlegungstag: 10. Dezember 1998

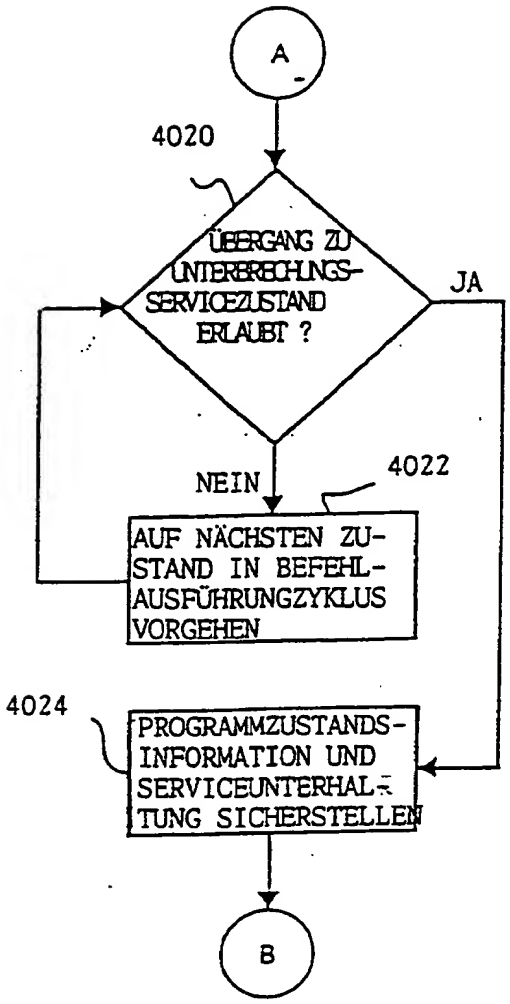


FIG. 25.B